Cadres formels pour la simulation des peuplements hétérogènes de plantes en compétition pour les ressources

Vincent Le Chevalier

ÉCOLE CENTRALE DES ARTS ET MANUFACTURES
« ÉCOLE CENTRALE PARIS »

Thèse
présentée par Vincent LE CHEVALIER
pour l'obtention du GRADE DE DOCTEUR

Spécialité : Mathématiques Appliquées
Laboratoire d'accueil : Mathématiques Appliquées aux Systèmes (MAS)

Cadres formels pour la simulation des peuplements hétérogènes de plantes en compétition pour les ressources

Soutenue le 19/05/2010
Devant un jury composé de :

M. Jean-Pierre MÜLLER	(Rapporteur)
M. Éric RAMAT	(Rapporteur)
M. Daniel AUCLAIR	
M. Jacques GIGNOUX	
M. Philippe de REFFYE	(Président)
M. Paul-Henry COURNÈDE	(Directeur)
M. Marc JAEGER	(Directeur)

N° ordre : 2010 ECAP 0010

Formal frameworks for the simulation of heterogenous plant populations competing for resources

This PhD is part of the Digiplant project, a multi-disciplinary effort between China and France on the topic of modelling, simulation and visualisation of plant growth. The goal of the PhD was to simulate the interactions between the GreenLab plant model and its environment in heterogenous landscapes. We especially wanted to model, simulate and visualise the interactions between plants and ressources and specifically water. The main achievements of the PhD were a simulation formalism and associated software architecture, a generic model of competition for resources, and a reformulation of the plant model which opens up many subsequent development opportunities.

Keywords: simulation, greenlab, landscape, competition, resources, synchronisation

Cette thèse prend place dans le cadre du projet DigiPlante, un effort de recherche pluridisciplinaire entre la Chine et la France sur les thèmes de modélisation, simulation et visualisation de la croissance des plantes. Le corps de la thèse concerne la simulation des interactions entre le modèle de plantes GreenLab et son environnement au sein de paysages hétérogènes. On cherche plus précisément à modéliser, simuler et visualiser l'interaction d'un ensemble de plantes avec les ressources, et en particulier avec les ressources en eau. La thèse a débouché principalement sur la conception d'une architecture de simulation et d'un modèle générique de compétition pour les ressources, et a aussi abouti à une reformulation du modèle de plantes lui-même qui ouvre de nombreuses pistes de développement.

Mots-clés : simulation, greenlab, paysage, compétition, ressources, synchronisation

REMERCIEMENTS

Je tiens ici à remercier en quelques lignes celles et ceux qui m'ont soutenu durant ces années de thèse.

Tout d'abord mon trio d'encadrants, Marc Jaeger, Paul-Henry Cournède et Philippe de Reffye, qui ont su se répartir la (lourde !) tâche de me diriger au fil de mes travaux. Marc a sans doute été le plus exposé à mes errements et à mon obstination et s'est montré d'une remarquable patience, apportant à chaque discussion des éclairages nouveaux qui m'ont permis d'avancer. Son recul sur les problématiques et son souci du détail m'ont été d'un grand secours chaque fois que j'ai rédigé une publication. Paul-Henry est celui qui m'a le premier permis de travailler dans l'équipe Digiplante en 2004 pour un projet d'élève, avant de me recruter pour effectuer cette thèse. Je le remercie pour la confiance qu'il m'a toujours accordée, pour sa disponibilité et pour l'intérêt incessant qu'il a témoigné aux évolutions parfois difficiles à anticiper de mes travaux. Philippe s'est toujours montré tolérant et encourageant au cours de cette thèse qui pourtant ne concernait que de loin ses propres intérêts de recherche, et ses explications sur le comportement des plantes ont été déterminantes pour certaines parties. Il m'a aussi communiqué un peu de la passion qui l'anime à chaque fois qu'il expose ses idées.

Je remercie mes deux rapporteurs, Éric Ramat et Jean-Pierre Müller, pour les critiques constructives qu'ils ont apporté, ainsi que Jacques Gignoux et Daniel Auclair pour l'intérêt qu'ils ont témoigné à mes travaux.

Je remercie le laboratoire MAS de l'École Centrale, qui m'a accueilli pendant ces années de thèse, sous la houlette de deux directeurs successifs, Christian Saguez et Étienne de Rocquigny. Je remercie également Jean-Hubert Schmitt, directeur de la recherche de Centrale, qui m'a donné les moyens financiers d'accomplir ce travail. Je tiens aussi à citer le CIRAD et l'UMR AMAP dont le support aux doctorants m'a permis de collaborer étroitement avec Marc Jaeger, et où j'ai à chaque fois reçu un excellent accueil. Le pôle de compétitivité Cap Digital (projet TerraNumerica), l'ANR Calcul Intensif et Simulation (projet 3Worlds) et la fondation Agropolis RTRA (projets « Plates-formes de simulation et de modélisation » et « Modélisation intégrative des paysages et écosystèmes ») ont participé au financement de mes travaux et je les en remercie.

Durant ma thèse j'ai fait de nombreux déplacements dont l'organisation aurait été très difficile sans l'aide précieuse de Sylvie Assens au Cirad, de Mélanie Broin au LISAH, de Christine Biard, Isabelle Biercewicz, Laura Norcy, Delphine Goyer et Martine Verneuille à l'INRIA, et de Hong Bizhen au Liama.

Au quotidien au laboratoire MAS, merci à Sylvie Dervin pour le défrichage de problèmes administratifs et l'humour surmontant les urgences et les poils de chat... Ainsi qu'à tous ceux dont j'ai partagé les bureaux, les pauses et les soirées pizzas-jeux : Sylvain, Marc, Cédric, Aurélie, Qi Rui, Takuya pour ne citer que le noyau dur, auxquels il faut rajouter l'ensemble des doctorants et assimilés des équipes Digiplante, MAS-Bio & friends.

Mes amis m'ont aussi beaucoup soutenu (et seulement un peu moqué de temps à autres) : merci à Dionoea, Tulkas, DiDjCodt, Yoann, Jix, Rem, Laug et JB en particulier pour la collocation de serveur qui s'est révélée très utile pour ma thèse et pour l'animation sur IRC... Les membres du club Mitsubachi m'ont aidé à évacuer la pression, avec mention spéciale à Christine et Thomas, les enseignants, pour la bonne ambiance qu'ils entretiennent savament.

Merci aussi à toute ma famille pour son soutien tant matériel que moral et en particulier à ma mère pour les corrections orthographiques dans le manuscrit. Mon grand-père et mon père sont ceux qui m'ont donné le goût des sciences, ce travail résulte en partie de ce qu'ils ont semé...

Et bien sûr, merci à Véro, qui mériterait d'apparaître en bonne place dans presque toutes les catégories précédentes et bien d'autres !

La mathématique est une science dangereuse : elle dévoile les supercheries et les erreurs de calcul.

Galileo Galilei

On two occasions, I have been asked [by members of Parliament], "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?" I am not able to rightly apprehend the kind of confusion of ideas that could provoke such a question.

Charles Babbage

The purpose of computing is insight, not numbers.

Richard Hamming

La perfection est atteinte, non pas lorsqu'il n'y a plus rien à ajouter, mais lorsqu'il n'y a plus rien à retirer.

Antoine de Saint-Exupéry

TABLE DES MATIÈRES

1. CONTEXTE, ENJEUX ET PROBLÉMATIQUES

Les plantes sont parmi les êtres vivants les plus indispensables à l'homme sur la planète. Elles fournissent de la nourriture, de l'énergie, et sont utilisées comme éléments de décoration dans des projets d'urbanisme. Elles participent aussi à de nombreux échanges de matière et d'énergie qui influent fortement sur le climat. La simulation de la croissance des plantes devient un enjeu fondamental pour la gestion de nos exploitations agricoles bien sûr, mais aussi de notre patrimoine forestier et paysager, voire à plus grande échelle pour la compréhension plus profonde des évolutions climatiques.

Cette thèse prend place dans le cadre du projet DigiPlante, un effort de recherche pluridisciplinaire sur les thèmes de modélisation, simulation et visualisation de la croissance des plantes. Le travail de thèse envisagé est l'exploration des possibilités du modèle de plantes structure-fonction GreenLab pour des applications à des peuplements hétérogènes en interaction avec l'environnement, et plus spécialement les ressources, à l'échelle du paysage. Cela signifie en particulier que l'on souhaite simuler non seulement l'effet de l'environnement sur les plantes, mais aussi l'action des plantes sur leur environnement. La compacité et l'efficience algorithmique du modèle GreenLab permettent d'envisager des simulations à grande échelle.

La décision de mener cette exploration résulte de plusieurs facteurs. D'abord, la maturité des modèles structure-fonction de plantes, dont le modèle GreenLab bien sûr. Mais surtout la constatation que beaucoup d'applications à venir devront s'intéresser à l'interaction avec l'environnement et fonctionner à l'échelle du paysage. Ne serait-ce qu'en terme de visualisation, il est nécessaire de dépasser l'échelle de la plante seule, et des travaux sur ce thème seront rappelés dans la section suivante. Les applications quantitatives prennent également tout leur sens au niveau du paysage en particulier pour l'aide à la décision. Enfin, l'étude des interactions avec l'environnement est incontournable pour assurer le caractère prédictif du modèle. Le travail a été très exploratoire, à cause de la nouveauté presque totale du champ d'étude dans les équipes impliquées.

Dans ce chapitre introductif, nous présenterons tout d'abord plus en détail le modèle GreenLab, puis d'autres éléments de contexte pour l'application aux paysages. Ensuite nous explorerons quelques problématiques particulières posées par cette thématique.

1.1 Contexte

1.1.1 Le modèle GreenLab

Historique

On peut considérer que l'histoire de GreenLab prend son origine dans la thèse de Phillipe de Reffye portant sur l'architecture des arbres et plus particulièrement axée sur l'étude du caféier [de Reffye(1979)]. Ce que l'on appelle ici architecture des arbres est la typologie et la topologie des axes, les angles les positionnant dans l'espace, et les positions des différents organes. Un nombre limité de critères ont été identfiés par les botanistes, comme le rythme de croissance, le positionnement des branches sur leur axe porteur ou la position des organes reproducteurs, et permettent de classifier les plantes suivant leur modèle architectural [Hallé et al.(1978)]. À ces notions s'ajoute celle de réitération, qui explique comment certaines branches peuvent adopter la même structure que la branche dont elles sont issues. Ces travaux ont permis le développement du logiciel AMAP, capable de créer des plantes virtuelles tridimensionnelles réalistes non seulement au sens visuel mais au sens botanique du terme [de Reffye et al.(1988), Jaeger et de Reffye(1992)]. Ces plantes virtuelles ont été utilisées depuis dans des contextes variés d'imagerie, et une start-up appelée Bionatics[1] assure leur valorisation commerciale.

À partir de cette base, deux principaux axes de recherche ont été poursuivis, l'un focalisé sur le fonctionnement de la plante et l'autre sur son développement, c'est-à-dire la mise en place de son architecture au fil du temps.

De nouveaux travaux botaniques ont permis de définir la notion d'âge physiologique [Barthélémy et Caraglio(2007)], qui donne une typologie des axes de la plantes. L'axe de référence décrit l'évolution de cette typologie le long des axes de croissance de la plante, ce qui inclut les ramifications mais aussi les mutations le long d'un axe. Le logiciel AMAP basait cette typologie exclusivement sur l'ordre de branchement, ce qui ne rend pas compte de toute la richesse du développement de la plante. Le modèle AMAPsim [Barczi et al.(1997)] remédiait à cet inconvénient, ouvrant ainsi la voie à des simulations plus réalistes du développement. Il est encore développé aujourd'hui et reste la référence pour ce qui est de la simulation du développement et de l'architecture des plantes.

Dans la nature, les dimensions des organes (feuilles, fruits, mais aussi tiges et branches) dépendent du fonctionnement même de la plante, en particulier de sa production de biomasse par photosynthèse et de la répartition de cette biomasse. Dans les approches précitées, ces dimensions sont au contraire fixées par celles des symboles tridimensionnels utilisés. Les logiciels AMAPpara [Blaise et de Reffye(1994)], puis AMAPhydro [de Reffye et al.(1999)] ont été développés pour intégrer à la fois le fonctionnement et le développement de la plante. Malheureusement, le temps de simulation en était fortement augmenté. Il fallait en effet simuler la photosynthèse, la circulation des assimilats dans les branches, l'allocation de la biomasse aux différents

[1] `http://www.bionatics.com/`

organes... Pour de grands arbres, les temps de calcul devenaient prohibitifs.

C'est à ce stade que le développement du modèle GreenLab commença. Il s'agit d'une formalisation mathématique d'AMAPhydro, basée sur des hypothèses simplificatrices qui accélèrent les calculs [de Reffye et Hu(2003), Cournède et al.(2006)], et intégrant certaines des avancées d'AMAPsim sur le plan du développement. Cette formalisation a multiplié les possibilités de modélisation. Par exemple, il devenait enfin possible de résoudre le problème inverse, c'est-à-dire de retrouver les paramètres du modèle permettant de reproduire des observations sur des plantes réelles. On pouvait également envisager des applications de la théorie du contrôle optimal aux plantes. Le domaine d'application passait de l'imagerie et de la visualisation à l'agronomie et à la foresterie. Ceci est rendu possible par l'influence historique d'AMAP et des botanistes sur le développement du modèle. Il a été depuis décliné en plusieurs versions, toutes fidèles à certains principes de base. Ces principes et ces versions sont décrits dans les deux sections suivantes.

Principes

Le, ou plutôt les modèles GreenLab sont des modèles structure-fonction, c'est-à-dire qu'ils simulent à la fois l'évolution de la structure de la plante (le développement, parfois appelé organogenèse) et le fonctionnement, l'ensemble des processus biophysiques contribuant aux variations de masse des organes [Sievänen et al.(2000), Prusinkiewicz(2004)]. En ce sens ils sont plus riches que les *process based models* qui ne simulent que le fonctionnement et se placent le plus souvent à l'échelle du couvert végétal. Ils visent à la généricité en s'attardant sur des caractéristiques et processus communs à toutes les plantes. Cependant, des hypothèses simplificatrices sont faites pour rendre les modèles plus compacts tout en conservant leur efficacité.

Tout d'abord, dans les modèles GreenLab, l'échelle du modèle se situe entre celle de la plante entière et celle de l'organe : l'unité de base de construction de l'architecture est le phytomère, c'est-à-dire l'entité botanique constituée d'un entre-noeud, du noeud situé à son extrémité, ainsi que des feuilles et des bourgeons portés par ce noeud [White(1979)]. On ne considère pas les processus détaillés de photosynthèse au sein des feuilles, par exemple, mais on fait juste un bilan global de production de biomasse et cette production est répartie aux organes. GreenLab est basé sur un modèle sources-puits d'allocation du carbone, qui tire parti du concept de *pool commun* (la biomasse produite est disponible instantanément pour tous les organes de la plante, il n'y a pas de diffusion à travers la plante [Heuvelink(1995)]).

La structure est très simplifiée. La position spatiale des organes n'intervient pas dans le modèle GreenLab, même si elle peut facilement être obtenue après coup pour des visualisations réalistes. Il n'y a pas de calcul détaillé d'interception lumineuse (remplacé par une application de la loi de Beer-Lambert dans les versions les plus récentes), ni de modélisation précise des racines. En fait seule la topologie et le nombre des organes interviennent dans le modèle.

Le modèle est temporellement discret, le pas de temps est le cycle de développement [de Reffye et Hu(2003)] qui est la durée entre l'apparition de deux unités de croissance

successives, par exemple un an pour des arbres en zone tempérée.

Ces hypothèses simplificatrices permettent de réduire drastiquement la complexité algorithmique du modèle, au point que l'on peut écrire une équation explicite donnant la production de biomasse à un cyle donné en fonction de la production aux cycles précédents. Cette compacité de l'écriture du modèle a été décrite comme une « re-mathématisation » des modèles de plantes [Varenne(2003)]. Cette dénomination est à mon avis un peu abusive vis-à-vis des modèles précédents, qui même s'ils étaient plus complexes algorithmiquement n'en étaient pas moins mathématiques eux aussi. Ceci sous-entend également qu'un modèle mathématique formellement compact est plus simple à manipuler, ce qui n'est pas évident (par exemple les équations de Navier-Stokes constituent un formalisme compact difficile à étudier et simuler).

On pourrait être inquiet de l'impact de toutes ces simplifications sur le réalisme du modèle de plante. Heureusement, les concepteurs des modèles GreenLab ont toujours eu un grand souci des applications quantitatives, et l'étude de la calibration du modèle est un souci permanent. Les ajustements de paramètres réalisés pour de nombreuses espèces (*Arabidopsis thaliana* [Christophe et al.(2008)], plantes agronomiques telles que le maïs [Guo et al.(2006), Ma et al.(2008)] ou la tomate [Dong et al.(2003), Dong et al.(2008)] et même arbres comme le hêtre [Letort et al.(2008)] ou le pin [Guo et al.(2007)]) prouvent que les modèles sont suffisamment puissants et pragmatiques pour analyser la croissance de plantes réelles. Cette visée applicative, avec la construction de modèles qui ne sont pas seulement conceptuels mais qui peuvent fournir des résultats réalistes, est une constante à travers toutes les versions de GreenLab.

Versions

La première version du modèle GreenLab, baptisée ultérieurement GL1, est caractérisée par un fonctionnement entièrement déterministe et l'absence de rétroaction du fonctionnement sur le développement. On spécifie ainsi un automate déterministe qui génère l'architecture de la plante, et des paramètres de fonctionnement qui permettent d'obtenir la masse des organes. La structure de la plante est factorisée en sous-structures identiques répétées dans la plante, augmentant ainsi l'efficience algorithmique du modèle [Yan et al.(2004)]. Cette version est bien adaptée aux plantes à l'architecture peu plastique, comme par exemple le tournesol ou le maïs. C'est sans doute encore la version la plus utilisée de par sa simplicité.

La deuxième version, GL2, introduit un processus de développement stochastique [Kang et al.(2004)]. Ceci permet de simuler la variabilité architecturale au sein d'une même espèce. En revanche, ceci complique à la fois la simulation et l'ajustement paramétrique, car trouver les paramètres de l'automate stochastique nécessite d'emblée plus d'observations.

Le modèle GL3 revient à un fonctionnement déterministe, mais introduit une rétroaction du fonctionnement sur le développement [Mathieu(2006), Mathieu et al.(2009)]. Ainsi le nombre d'organes généré et la topologie dépendent d'un indicateur de la compétition trophique interne à la plante (rapport entre la source et les puits). Ceci permet

de modéliser la plasticité architecturale des plantes en fonction des conditions environnementales, et accessoirement de générer des architectures quasiment aussi variées que celles produites par GL2. Cette version du modèle suscite de plus en plus d'intérêt, même si la calibration est difficile.

Le modèle GL4 représente la fusion de GL2 et GL3. Les probabilités associées à l'automate stochastique dépendraient ainsi de l'état trophique de la plante. Cette version ne peut pas encore être considérée comme achevée, le formalisme même de GL4 n'étant pas à ce jour dans un état stable (le plus proche à ce jour est décrit dans [Pallas et al.(2009)]).

Enfin, la dernière version en développement de GreenLab est GL5. Elle devrait permettre de simuler des phénomènes particuliers se déroulant à différentes phases du cycle de développement (préformation et néoformation, polycyclisme). Cette version est encore en développement à l'heure actuelle.

Implémentations

Il y a deux implémentations dominantes des modèles GreenLab à l'heure actuelle.

La première est le logiciel open-source GreenScilab[2] (en fait une *toolbox* pour Scilab), développé principalement au Liama (Laboratoire franco-chinois de recherche en Informatique, Automatique et Mathématiques Appliquées) en collaboration avec l'INRIA et l'École Centrale Paris. Il est codé dans le langage Scilab et implémente les versions GL1 et GL2. C'est un outil très efficace pour répandre la connaissance du modèle GreenLab, de par sa licence. Il a servi, par exemple, à plusieurs tutoriaux sur le modèle.

La seconde implémentation est le logiciel Digiplante, développé à l'École Centrale Paris et à l'INRIA. C'est un logiciel propriétaire codé en C++. Il est plus performant que GreenScilab, doté de capacité de visualisation 3D plus avancées, et est très orienté vers la calibration du modèle. Il ne fonctionne à ce jour qu'avec les versions déterministes GL1 et GL3. C'est ce logiciel qui jusqu'à maintenant a servi à la plupart des applications.

Plus récement de nouvelles implémentations commencent à voir le jour, en particulier QingYuan au Liama, et Gloups, l'implémentation de GL5 réalisée au Cirad. Et bien sûr certains doctorants développent leurs propres codes suivant leurs besoins spécifiques ; mon implémentation personnelle d'une nouvelle version de GreenLab, rendue nécessaire par les contraintes du couplage avec d'autres modèles, est ainsi décrite au chapitre 5.

1.1.2 Travaux de pure visualisation

Comme nous l'avons vu dans le cadre des modèles de plantes, une des premières motivation pour le travail au niveau du paysage a été la synthèse d'images réalistes.

[2] `http://www.scilab.org/products/modules/pem/greenscilab`

Dans un sens, c'est un domaine scientifiquement moins exigeant, ne nécessitant pas de coûteuses mesures sur le terrain et n'imposant pas de calibration des modèles. Par contre, le défi technique peut être immense, en particulier si des performances « temps réel » sont recherchées. Les contributions des spécialistes de ce domaine ont été pour nous une importante source d'inspiration.

Une étude pionnière dans ce domaine est celle de J. Hammes [Hammes(2001)] qui a introduit pour la première fois la notion d'écosystème dans le domaine du graphisme sur ordinateur. Il proposait une approche basée sur des textures multi-niveaux pour synthétiser un paysage visuellement convaincant.

La génération du relief a aussi été un domaine de recherche particulièrement actif [Fournier et al.(1982), Zhou et al.(2007)]. Ceci inclut par exemple des études sur la simulation de l'érosion, avec lesquelles j'avais d'ailleurs eu l'occasion de me familiariser avant de commencer ma thèse [Musgrave et al.(1989), Chiba et al.(1998), Neidhold et al.(2005)]. Ce domaine est encore très actif aujourd'hui, avec en particulier une utilisation de plus en plus poussée des capacités des nouvelles cartes graphiques [Schneider et al.(2006), Dachsbacher(2006)]. Cette focalisation sur le relief se comprend aisément dans un cadre où il importe de créer une ambiance visuelle ; un terrain réaliste, éclairé soigneusement avec des effets atmosphériques tels que le brouillard ou les nuages donne une illusion très convaincante même avec des textures très basiques, tant que la distance d'observation est assez grande.

À une distance plus rapprochée les objets du paysage deviennent importants, et en particulier les plantes. La chaîne graphique AMAP, pourvue d'un éditeur de scène, permet de représenter des peuplements de plusieurs dizaines d'individus mais n'offre pas la possibilité d'une extension au niveau du paysage. En 2007, la société Bionatics a lancé une nouvelle application, LandSim3D, qui intègre des plantes basées sur les modèles 3D AMAP dont le niveau de détail se dégrade en fonction de la distance. D'autres auteurs ont généré des modèles de plantes « poussant » dans des espaces voxels [Greene(1989), Blaise et al.(1998)], d'autres encore ont utilisé des L-Systems [Deussen et al.(1998)]. Certaines de ces approches intègrent même certaines idées de compétition spatiale pour générer les positions des plantes de manière plus réaliste [Alweis et Deussen(2005), Mech et Prusinkiewicz(1996)]. D'autres auteurs étudient aussi l'impact des végétaux sur d'autres phénomènes visuellement importants, tel l'érosion [Wang et Hu(2009)].

Par rapport à nos objectifs, toutes ces approches pèchent principalement par le manque de rétroaction entre les éléments du paysage, ce qui mène à des comportements dynamiques insuffisament réalistes. Généralement le but est d'obtenir un état statique réaliste du paysage dans lequel sont rajoutés par la suite des phénomènes à très petite échelle de temps. Par exemple, on génère une forêt, puis on fait en sorte de simuler l'effet du vent pour faire une animation, mais on ne voit pas la forêt pousser. Le réalisme physique n'est recherché que dans la mesure où il permet une modélisation (au sens de l'obtention d'un modèle 3D) plus aisée.

1.1.3 Modélisation quantitative

D'autres chercheurs poursuivent cependant des objectifs plus quantitatifs avec une modélisation à l'échelle du paysage.

La modélisation hydrologique en est un bon exemple. On peut citer en exemple le modèle SHETRAN [Ewen et al.(2000)], à l'échelle du bassin versant, qui a l'ambition d'intégrer de nombreux phénomènes et en particulier le ruissellement de surface et le transfert d'eau dans le sol lui-même. Le ruissellement de surface est particulièrement étudié [Taddei et al.(2000), Liu et al.(2006)] puisque cela permet en particulier d'espérer modéliser les crues dues à des évènements importants de précipitation. Ceci dit, un souci persistant avec des modèles à cette échelle est celui de la validation [Bormann et al.(2007)]. Il est difficile d'obtenir des données, difficile de mener des expérimentations... Donc difficile de valider, d'infirmer et de comparer les modèles.

Un autre domaine lié est l'étude de l'érosion, que nous avons déjà mentionné dans la section sur la visualisation. Ce phénomène est intéressant car il peut être étudié à deux échelles de temps différentes dans des optiques différentes. Les géomorphologistes le simulent car c'est l'un des processus déterminant la structure des massifs montagneux [Howard(1994)]. À une telle échelle de temps, on a plus affaire à un modèle explicatif qui décrit les phénomènes en jeu sans beaucoup d'ambitions prédictives. Mais d'autres études comme [Mitasova et al.(1996)] s'intéressent aux conséquences plus immédiates de l'érosion sur les capacités des sols pour l'agriculture.

Bien entendu il ne faut pas négliger non plus les modèles à des échelles spatiales inférieures, dont on peut se servir pour construire un modèle de paysage. Nous avons déjà décrit un modèle de plante individuelle existant, mais d'autres modèles fonctionnent aussi à l'échelle de la parcelle et intègrent un modèle de sol, par exemple [Mailhol et al.(1997)]. La science du sol est aussi bien développée depuis longtemps [Kosugi et al.(2002)], et les modèles détaillés de circulation d'eau intègrent parfois une modélisation des racines [Varado et al.(2005), Vrugt et al.(2001)].

Dans tous ces modèles, la modélisation du fonctionnement des plantes est bien moins détaillée que ce que l'on peut obtenir avec GreenLab. Dans le travail de la thèse, on a donc cherché à simplifier au maximum les éléments externes à la plante mais en conservant la richesse du modèle de plante lui-même.

1.2 Problématiques

Cette section contient une description non-exhaustive des problématiques soulevées par la thématique de recherche.

1.2.1 Définition et structure du paysage

La définition même de ce que l'on appelle paysage est difficile. Suivant les communautés, l'échelle spatiale, l'échelle temporelle, l'extension géographique, les phénomènes considérés varient. Comme c'est un terme du langage courant, les auteurs

s'en remettent souvent à la compréhension intuitive du lecteur sans formaliser la définition (par exemple il n'y a pas de définition explicite du paysage dans le livre [Costanza et Voinov(2004)], pourtant assez générique).

Au sens premier, un paysage est ce qui s'offre à la vue. Les paysagistes utilisent encore le terme dans ce sens. Cette définition implique l'existence d'un point de vue, et compte-tenu des capacités limitées de l'être humain pose d'emblée des limites en étendue et en résolution. L'étendue spatiale se retrouve alors restreinte à la distance de l'horizon si les conditions atmosphériques le permettent. Cette distance est donnée par :

$$d = \sqrt{(R+h)^2 - R^2} \tag{1.1}$$

où h est la hauteur de l'œil, R est le rayon terrestre, et d la distance de l'horizon. Une application numérique donne $d \approx 5km$ pour $h = 2m$, et $d \approx 230km$ pour $h = 4000m$ (le Mont-Blanc). Par ce calcul très simple on trouve une définition assez intuitive de l'étendue spatiale d'un paysage et qui correspond finalement bien à l'utilisation implicite du terme dans de nombreux contextes. La précision de la description du paysage, elle, va dépendre beaucoup de l'application ; pour de la visualisation pure elle peut même être variable suivant la distance au point de vue.

Cette définition se caractérise par son côté anthropocentrique, car elle utilise les limitations de la vision d'un être humain. D'autres approches, particulièrement en écologie du paysage [Dunning et al.(1992)], suppriment cette limitation en se centrant sur un organisme quelconque. Ainsi le paysage d'un faucon et celui d'une fourmi ne couvrent pas la même étendue spatiale. Dans ce cas là, la notion déterminante est celle d'hétérogénéité, le paysage devenant une étendue spatiale pouvant être décrite par des parcelles distinctes. Nous avons préféré conserver une définition plus restrictive en fixant l'étendue.

Même cette notion de paysage impose pratiquement d'emblée une notion d'hétérogénéité. À l'exception de grandes étendues désertiques, ces ordres de grandeurs pour l'extension géographique ne se prêtent pas à une description uniforme, que ce soit en termes d'aspect visuel ou en termes de fonctionnement. Une portion relativement uniforme de paysage (une prairie, par exemple) est mieux désignée par le terme de *parcelle* (et souvent cela correspond à une simplification, car même une prairie n'est pas homogène). D'un autre côté, une région entière, un pays, un continent ne sont pas naturellement qualifiés de paysage. Je pense que cela est dû au problème de définition précédement décrit : un paysage doit pouvoir s'appréhender par la vue, au moins potentiellement.

Stephen M. Ervin [Ervin et Hasbrouck(2001)] donne un autre point de vue sur l'hétérogénéité du paysage en le décomposant en éléments :

- Terrain
- Végétation
- Eau
- Constructions
- Animaux (incluant l'être humain)

– Atmosphère (incluant l'éclairage)

Ce sont les interactions de ces éléments qui déterminent *in fine* l'aspect visuel du paysage. On voit que cette décomposition implique également une hétérogénéité fonctionnelle, l'évolution de chacun des éléments ne se prêtant pas aux mêmes modélisations. Il est important de noter que l'aspect final du paysage est un résultat de son évolution dynamique, de même que l'aspect d'une plante à un instant donné nous renseigne en partie sur l'historique de sa croissance. Des approches purement descriptives, telles celles adoptées par les SIG (Systèmes d'Information Géographiques), sont donc limitées, même si elles peuvent bien sûr servir de support à la modélisation dynamique du paysage.

Quand on prend de la distance par rapport au rendu visuel, on peut être amené à modifier les définitions de l'étendue spatiale du paysage. Par exemple, l'objet d'étude des hydrologues est le bassin versant [Campy et Macaire(2003), p.72], qui est une notion indépendante de toute possibilité d'appréhension visuelle. Ceci étant, on peut retrouver le lien avec un certain point de vue : le bassin versant n'est-il pas l'étendue spatiale de terrain pouvant contribuer à l'écoulement d'eau au point de sortie ? Remplaçons ce point par l'œil, l'eau par la lumière, et nous retrouvons la définition initiale... Et cependant, on sera plus enclin à désigner comme paysage un bassin versant de dimension modeste, plutôt que celui d'un fleuve entier. Même du point de vue fonctionnel, il semble que ce soient les limites physiques de notre vision qui fixent l'étendue spatiale maximale considérée comme paysage.

La limite inférieure en étendue à partir de laquelle on ne parle plus de paysage est également intéressante. Un jardin peut-il être considéré comme un paysage ? L'intérieur d'un bâtiment ? Je pense personnellement qu'on ne considère comme paysage que des étendues spatiales qu'on ne peut pas parcourir facilement à pied. Cela veut dire que notre appréhension du paysage est presque uniquement visuelle (et intellectuelle bien sûr). Il n'est pas possible de toucher un paysage, alors que l'on peut se promener dans un jardin et agir directement, mécaniquement sur ses composants, en un temps très bref. Ceci peut aussi fournir une définition alternative de la parcelle, qui elle peut se parcourir physiquement.

Voici donc la définition du paysage que nous avons adoptée (implicitement au début) pour l'étude : c'est une portion de la surface terrestre, d'une étendue comprise en ordre de grandeur entre le kilomètre et la centaine de kilomètres, sur laquelle interagissent plusieurs composants physiques et biologiques, étudiée depuis un certain point de vue qu'il soit visuel ou fonctionnel.

On a préféré cette définition à une autre faisant davantage intervenir la notion d'écosystème. Ainsi un paysage au sens où nous l'entendons peut couvrir une partie d'un écosystème ou être le lieu d'interaction de plusieurs écosystèmes. Il va de soi que cette définition n'a pas de vocation universelle et n'invalide pas les autres définitions que l'on pourrait trouver. Mais elle nous permet de poser clairement notre objet d'étude.

1.2.2 Vers un « paysage fonctionnel » ?

Le terme de « paysage fonctionnel » a été utilisé par Marc Jaeger par analogie avec l'imagerie médicale, et désigne une simulation et visualisation d'un paysage montrant au minimum des dynamiques qualitativement réalistes [Jaeger et Teng(2003)]. Le terme est un peu abusif et plus restrictif que ce que l'on pourrait attendre. Cet objectif est à comprendre comme un parallèle avec le passage entre AMAP et AMAPsim ou AMAPhydro. L'ajout d'une évolution dynamique permet non seulement d'obtenir plus facilement un aspect réaliste, mais est aussi le premier pas vers des applications quantitatives.

Impact visuel du fonctionnement

Comme il a été dit plus haut, l'aspect visuel d'un paysage, comme celui d'une plante, dépend en réalité de toute la dynamique de son évolution. Même si l'objet final peut être très complexe, les phénomènes qui le construisent peuvent être simples, et donc en modélisant ces phénomènes on peut obtenir une façon plus commode de parvenir à un aspect réaliste. En outre la représentation peut se révéler plus compacte car il est possible de stocker uniquement les données initiales et les paramètres des processus, plutôt que le résultat final.

Dans un sens, cette partie de l'approche est une forme de modélisation procédurale [Ebert et al.(2002)]. Ce procédé est capable de générer des modèles très convaincants et complexes en utilisant des algorithmes, au lieu de spécifier explicitement tous les détails. Par exemple pour le cas d'une texture, l'approche explicite consiste en l'utilisation d'une image qui sera plaquée sur le modèle 3D. L'approche procédurale spécifie plutôt une fonction donnant la couleur en chaque point de l'espace, et c'est le moteur de rendu qui appelle cette fonction autant qu'il est besoin. Ce genre d'approche a d'ailleurs été utilisé très souvent pour produire des paysages ou même des planètes entières, voir encore les exemples dans [Ebert et al.(2002), chapitre 20].

La différence est que l'approche procédurale ne sous-entend pas forcément une dynamique réaliste, mais recherche simplement un algorithme permettant d'arriver au résultat souhaité. Il se peut que l'on trouve des modèles simples permettant de générer un résultat proche du régime stationnaire atteint par un système dynamique plus complexe, ou similaire à l'état atteint à un instant donné. Par exemple, des modèles dynamiques complexes ont été réalisés pour rendre compte au mieux des phénomènes de brousse tigrée [Gilad et al.(2007)]. Ces motifs sont formés par la végétation en zones arides, faisant apparaître des zones de sol nu alternant avec des zones couvertes, et résultent d'une interaction complexe entre la dynamique du peuplement végétal, la pente du terrain et les précipitations. Mais le motif formé à un instant donné peut être décrit de manière bien plus synthétique, formant suivant les cas des taches, des zébrures, des labyrinthes... Cependant certaines approches procédurales peuvent se fonder sur des dynamiques réalistes, car c'est souvent une façon très efficace, stable et souple de parvenir à de bons résultats.

Applications quantitatives

Il va de soi que des applications quantitatives nécessitent une modélisation des dynamiques et du fonctionnement même du paysage, au-delà de l'illusion visuelle. Dans le contexte du changement climatique, des systèmes d'aide à la décision capables de prendre pleinement en compte l'aspect biophysique vont devenir indispensables.

Là encore on peut faire le parallèle avec les plantes. Quand il s'agit de générer un modèle 3D d'arbre jugé convaincant par la majeure partie du public, une approche type AMAP est amplement suffisante (on peut arguer qu'elle est trop détaillée compte tenu des connaissances botaniques du public, ainsi des logiciels tels XFrog[3] ou Onyx[4]) sont largement utilisés bien que non fondés sur la botanique). Quand on cherche l'optimisation des apports en eau, ou l'étude des variations environnementales qui peuvent être déduites de l'état des arbres, un modèle de type GreenLab devient nécessaire.

Il convient cependant de noter qu'à ce stade de notre étude, nous n'avons pas l'ambition d'atteindre le même niveau de fidélité que le modèle GreenLab à l'échelle du paysage... Mais l'inclusion du fonctionnement constitue un premier pas nécessaire dans cette direction. Compte-tenu de l'hétérogénéité fonctionnelle des paysage, le chemin naturel de notre point de vue pour inclure le fonctionnement est celui de la modélisation intégrative.

1.2.3 Modélisation intégrative

La *modélisation intégrative* est une méthodologie de modélisation qui consiste à considérer un système complexe comme composé de plusieurs sous-systèmes couplés. Ces sous-systèmes sont modélisés séparément (éventuellement récursivement de façon intégrative) et ce sont les propriétés émergentes du couplage qui rendent compte de toute la richesse de comportement du système complet.

Cette démarche est naturellement attirante pour l'étude du paysage, car elle correspond à l'hétérogénéité fonctionnelle que nous avons mentionnée plus haut. Ainsi, une démarche intuitive est de modéliser chacun des composants du paysage, et de coupler le tout pour obtenir un modèle complet du paysage. Cela permet d'envisager la réutilisation de connaissances et modèles conçus par des spécialistes de chacun des éléments, et donc de se baser sur une grande quantité de travaux existants et validés. De plus, des modèles intégratifs sont souvent perçus comme plus fortement explicatifs car ils montrent une décomposition en sous-systèmes plus simples.

En l'occurrence dans le cadre de notre étude, comme nous voulions nous baser sur un modèle de plante existant, une démarche de modélisation intégrative s'imposait quasiment d'elle même, dans le sens où en isolant les plantes nous avions déjà entamé la décomposition du paysage en sous-modèles.

Ceci dit, la modélisation intégrative dans le cas des systèmes complexes naturels soulève certaines inquiétudes légitimes, en particulier en terme de validation. Dans

[3] `http://www.xfrog.com/`
[4] `http://www.onyxtree.com/`

des systèmes qui n'ont pas été conçus par l'homme, il est difficile d'être certain des modalités exactes du couplage ou du degré d'indépendance des sous-modèles. Or si le couplage est faux, quel que soit le degré de validité de chacun des sous-modèles, le comportement du système global sera faux lui aussi... Dans le même ordre d'idée, coupler des modèles qui font des hypothèses différentes en terme de gamme de validité de leurs entrées et sorties, ou des modèles non indépendants (par exemple plantes et sol+plantes) n'est pas sans risque. Cela veut dire que pour des applications quantitatives, il n'est pas raisonnable de se passer d'une étape de validation du système complet, quelle que soit la fiabilité des sous-modèles.

1.2.4 Échelles et modélisation

Le terme de multi-échelle revient souvent dans le contexte de l'étude des systèmes complexes comme le paysage, cependant la définition précise de ce qu'est l'échelle est assez difficile et nous allons voir que le concept de multi-échelle peut se prêter à diverses interprétations.

Une description générale des concepts d'échelle tels qu'ils s'appliquent au paysage peut être trouvée dans l'article [Turner et al.(1989)]. Mais il paraît utile de faire ici une analyse de ces problèmes centrée davantage sur leur impact dans un processus de modélisation intégrative. Il faut noter qu'une bonne partie des idées développées ici ne nous sont apparues clairement qu'à la fin de la thèse, et n'ont donc malheureusement pas toujours été rigoureusement suivies dans les faits...

Échelle d'étude d'un signal

La base de la modélisation est de chercher un lien explicatif entre des données d'entrée et des données de sortie, qui peuvent être considérées comme des échantillonages de signaux d'entrée et de sortie. C'est la description et la mesure de ces signaux qui va fixer les échelles du modèle, avant même de considérer le mode de calcul du modèle. En effet, lors de l'observation d'un signal, on se trouve confronté à des contraintes d'échantillonnage, d'étendue d'observation (durée dans le domaine temporel), de bruit et d'incertitudes de mesure, qui forcent certains choix de modélisation. Je pense qu'une interprétation dans le domaine fréquentiel (ou de manière équivalente dans celui des périodes) est la plus puissante. Le passage d'un domaine à l'autre est bien formalisé, et de nombreux travaux de traitement du signal ont posé des concepts fondamentaux en lien avec la théorie de l'information [Max(1987)].

Quelque soit le signal on choisit toujours une fréquence maximale observable. En général ce sera la fréquence des observations que l'on va faire pour valider le modèle. Quelquefois par choix de modélisation on va volontairement choisir une fréquence maximale assez faible si on ne cherche à expliquer que les grandes tendances. Ce choix de la fréquence maximale a des conséquences sur les observations des basses fréquences, à cause des phénomènes d'aliasing en particulier, et aussi parce que l'on va chercher à extrapoler les basses fréquences sur une étendue plus large. Conjugué au bruit et au temps d'observation forcément fini, cela veut dire que l'on fixe une autre

borne, une fréquence minimale en deçà de laquelle on ne considère que la composante constante du signal.

Finalement, l'échelle de modélisation d'un signal est définie par cette gamme de fréquence. La période minimale considéré fixe le *grain*, la période maximale *l'étendue* de l'étude. Bien sûr cette interprétation de la notion d'échelle est valable tant dans le domaine spatial que temporel. Un même signal peut être décrit à différentes échelles, en privilégiant par exemple les hautes fréquences (en général cela implique un échantillonnage serré sur une courte étendue) ou les basses fréquences (échantillonnage plus espacé mais sur une étendue plus large). Ceci ne change pas le signal lui-même mais simplement la finesse de la représentation que l'on en a.

Échelles d'un modèle

Pour nous, les échelles d'un modèle sont la réunion des échelles de ses entrées et de ses sorties. Ce sont les échelles auxquelles le modèle est validé ; moyennant un signal d'entrée à la bonne échelle, le modèle produit un signal de sortie qui, quand il est étudié à la même échelle que des données réelles, les reproduit convenablement.

Par exemple pour le modèle GreenLab, les échelles temporelles d'entrée et de sortie sont identiques : le grain est la durée du cycle de développement de la plante, l'étendue est la durée de vie de la plante. Spatialement, le grain est l'organe, l'étendue est la plante entière.

Certains modèles ont des échelles internes, en particulier les modèles de simulation qui sont en général discrétisés. Cependant, les modèles mathématiques comme les équations aux dérivées partielles n'en ont pas, même si les échelles d'espace et de temps sont généralement liées dans ces modèles on peut tout de même choisir abritrairement l'une d'elle. Qui plus est, l'échelle interne d'un modèle peut être différente de son échelle de validité ; ainsi une EDP pourrait décrire l'évolution à grande échelle d'un système, et cependant nécessiter une simulation numérique à un grain fin pour être précise. La plupart des modèles dynamiques ne sont pas limités en étendue, du moins en théorie, et donc peuvent être extrapolés bien au-delà de leur étendue de validité.

Autant il peut être relativement facile de changer l'échelle interne, de calcul, d'un modèle, autant il est difficile de changer ses échelles de validité en entrée et en sortie, car cela nécessite généralement des mesures supplémentaires. Un exemple de changement d'échelle interne, non validé par de nouvelles mesures jusqu'ici, se trouve dans le chapitre 5 exposant une nouvelle variante du modèle GreenLab (le passage à des équations continues permettant une discrétisation temporelle arbitraire). Mon opinion est que les échelles d'étude des signaux d'entrée et de sortie définissent mieux l'échelle d'un modèle que son éventuelle discrétisation numérique interne, et que en l'occurence on ne peut pas encore prétendre avoir changé l'échelle du modèle GreenLab.

Sens du terme « multi-échelle »

Lorsque l'on dit d'un modèle qu'il est multi-échelle, cela peut recouvrir plusieurs réalités différentes.

Au sens fort, un modèle multi-échelle fonctionne de manière identique et est validé à différentes échelles. C'est le cas par exemple des modèles qui génèrent des signaux fractals (l'article [Jackson et Streator(2006)] en donne un exemple pour des calculs mécaniques). Ce type de modèle est somme toute assez rare ; même les formes naturelles les plus proches de fractales ne le sont que sur une certaine gamme d'échelle. Un modèle multi-échelle au sens fort se définit par une large gamme de fréquence au niveau des signaux étudiés. Il faut noter que l'on peut appliquer la plupart des modèles comme s'ils étaient fortement multi-échelle, mais que c'est souvent à tort en l'absence de validation...

De nombreux phénomènes en particulier écologiques ne se modélisent pas de la même façon suivant l'échelle considérée, par exemple certaines corrélations entre des présences d'espèces apparaissent négatives à petite échelle et positives à grande échelle, comme décrit plus en détail dans [Wiens(1989)]. Cela montre que si le modèle associé à une échelle est appliqué à une autre comme si il était fortement multi-échelle, on peut faire de graves erreurs...

On peut obtenir un modèle multi-échelle au sens faible en faisant une collection de modèles différents d'un même phénomène à différentes échelles. En règle générale, cela ne se traduit pas par une gamme de fréquences étudiée élargie, mais plutôt par un ensemble de gammes de fréquences éventuellement disjointes. Un exemple en physique est la mécanique statistique, le passage entre loi des gaz parfaits et comportement individuel de molécules de gaz. En synthèse d'image, les modèles à niveaux de détail (LOD) [Clark(1976)] suivent le même principe. Le problème dans ce cas, outre la construction des modèles, est de choisir quand passer à l'échelle... Souvent cela induit des discontinuités dans les résultats. Cela est visible sur des images de synthèse, lorsque l'on voit les modèles 3D gagner ou perdre brusquement en détail à certaines distances. Ce problème est résolu spécifiquement pour les modèles de plantes pour la représentation du feuillage [Deng et al.(2010)]. Cette méthode de modélisation multi-échelle nécessite un effort de modélisation plus grand, mais au moins on peut espérer la validité sur une gamme d'échelle plus large donc une meilleure compatibilité avec d'autres modèles, ce qui est important dans un cadre de modélisation intégrative.

Multi-échelle, modélisation intégrative et optimisation des temps de calcul

On conçoit facilement que dans le cadre de la modélisation intégrative, la compatibilité entre les échelles des différents modèles devrait être un souci permanent. Coupler un modèle sortant des signaux valides à grande échelle avec un autre prenant en entrée des signaux à petite échelle, ou inversement, peut être numériquement possible (moyennant une modification des échelles de calcul) mais n'avoir scientifiquement aucun sens en l'absence de validation.

Il peut être tentant de conserver pour chaque modèle une échelle de calcul distincte, car cela économise des calculs : les modèles à grande échelle interne font des calculs moins fréquents. Mais cela impose une transformation de signaux à grande échelle en signaux à petite échelle utilisant l'une ou l'autre procédure d'interpolation, ce qui constitue en soi une hypothèse de modélisation qu'il faudrait valider (on choisit

en fait quelles hautes fréquences on va ajouter). Même si l'on peut éventuellement gagner du temps de calcul de cette façon on reste donc confronté au problème de la validation. Au final on a effectivement changé l'échelle d'un des modèles pour les faire fonctionner à une échelle commune, et il est bon que ce changement soit explicite et pas automatique.

À mon avis un formalisme de modélisation intégrative ne doit pas comprendre d'aspect multi-échelle autre que fonctionnel (l'imbrication des sous-systèmes), mais plutôt faire l'hypothèse que tous les modèles peuvent travailler à une échelle commune. Si les temps de calcul sont prohibitifs, il faut changer les modèles, ce qui peut inclure l'utilisation d'information sur les échelles et des méthodes d'intégration/interpolation. Mais le problème d'optimisation des temps de calcul gagne à être séparé des problèmes d'échelles car il peut aussi être résolu par le recours à d'autres techniques. Et en aucun cas on ne peut se passer d'une étape de validation si les compatibilités d'échelles ne sont pas respectées.

1.3 Méthodologie et axes de travail

Conscients que l'étude du paysage dans toute sa globalité était un projet beaucoup trop ambitieux, nous nous sommes focalisés sur les possibilités et les contraintes en matière de simulation seulement. C'est une étape obligatoire avant toute validation et application concrète. De plus, nous nous sommes volontairement restreints aux phénomènes biophysiques, excluant ainsi les problématiques sociales, politiques, économiques, etc. En fait notre principal intérêt s'est porté sur les ressources au sens de grandeurs physiques conservatives, en particulier l'eau, et la compétition pour celles-ci entre les plantes. Un des objectifs de la thèse était de vérifier la compatibilité du modèle GreenLab avec des simulations en environnement hétérogène à l'échelle du paysage.

Nous avons adopté une approche que l'on pourrait qualifier de « bottom-up », partant de ce que savions faire au sein de l'équipe, construisant des prototypes techniques puis finalement raffinant des approches plus formelles pour résoudre les problèmes détectés. Ceci nous a permis de réduire les développements et de conserver une certaine parcimonie de formalisme, en évitant la tentation de construire une plate-forme de simulation possédant des capacités bien supérieures à nos besoins propres. L'impact de cette méthodologie sur les développements que nous avons mené est décrit dans [Le Chevalier et Jaeger(2010)].

Dans le chapitre 2 on trouvera la description de nos premiers essais techniques, avec une analyse de leurs qualités et défauts. Ce sont ces deux premiers prototypes qui nous ont lancé sur les pistes de réflexion exploitées dans les chapitres suivants.

Tout d'abord, le chapitre 3 présente un formalisme de modélisation et l'architecture de simulation associée, qui permettent de résoudre de façon assez générique le problème de la synchronisation et le couplage de modèles qui se posait de manière frappante dans les prototypes. L'architecture proposée repose sur une séparation nette entre les données de la simulation et les composants de calculs, ce qui permet des simulations

plus stables et cohérentes.

Le cadre formel de modélisation se prête assez naturellement à la réalisation de modèles de compétition pour les ressources, comme décrit au chapitre 4. Ces modèles comblent une autre faille des prototypes : à cause de la faiblesse des architectures logicielles sous-jacentes, la simulation de la compétition pour les ressources était difficile et génératrice d'erreurs.

Malgré ses qualités, le formalisme de modélisation impose des contraintes aux modèles, qui méritent d'être attentivement considérées sous peine de perdre en réalisme. Cela peut imposer des efforts de modélisation supplémentaire. Le chapitre 5 présente ainsi une nouvelle version du modèle GreenLab, adaptée au cadre formel. Ce développement présente en outre certains intérêts même dans le cadre de la modélisation de la plante seule et, vu le contexte de la thèse, mérite que l'on s'y attarde.

Enfin, au chapitre 6 on détaille plusieurs simulations illustrant les possibilités de l'approche et des modèles développés. Seront ainsi étudiés le couplage du modèle de plante avec la ressource en eau, la gestion de la spatialisation, et un cas simple de peuplement hétérogène. Ces illustrations pourraient former la base d'un nouveau simulateur de « paysage fonctionnel »...

2. PROTOTYPES DE SIMULATEUR

Dans ce chapitre, nous donnons la description de deux prototypes techniques de « paysages fonctionnels ». Ce sont les enseignements tirés de ces deux prototypes qui ont conduit aux développements plus théoriques exposés aux chapitres 3, 4 et 5.

2.1 Premier prototype de « paysage fonctionnel »

Le tout premier prototype de « paysage fonctionnel » a été développé au Liama essentiellement par Aurélien Lesluye et Marc Jaeger. Il capitalisait sur des travaux précédents d'imagerie volumique [Kaufmann(1991)] appliqués tant dans un contexte médical que pour la visualisation de plantes [Jaeger et Teng(2003)].

Une description de ce prototype a été publiée dans [Le Chevalier et al.(2007b)].

2.1.1 Description

Le but premier de ce prototype était de tester des approches techniques qui pourraient permettre de spatialiser le modèle GreenLab dans un contexte paysage. Le réalisme des modèles et de leur implémentation était donc un souci secondaire.

L'ensemble du prototype est basé sur une discrétisation voxel.

Modèles

Ce prototype était focalisé sur la simulation du cycle de l'eau en relation avec le modèle de plante. Ce choix a été guidé par l'importance primordiale de l'eau comme ressource limitante dans la croissance de la plante. Ce cycle est partitionné en 5 phases :

- précipitations
- ruissellement et absorption par le sol
- diffusion dans le sol
- transpiration par les plantes
- évaporation à la surface du sol

Les mouvements d'eau associés à chaque phase sont schématisés sur la figure 2.1. Chaque phase fait appel à un modèle spécifique avec éventuellement une interprétation différente de l'espace voxel.

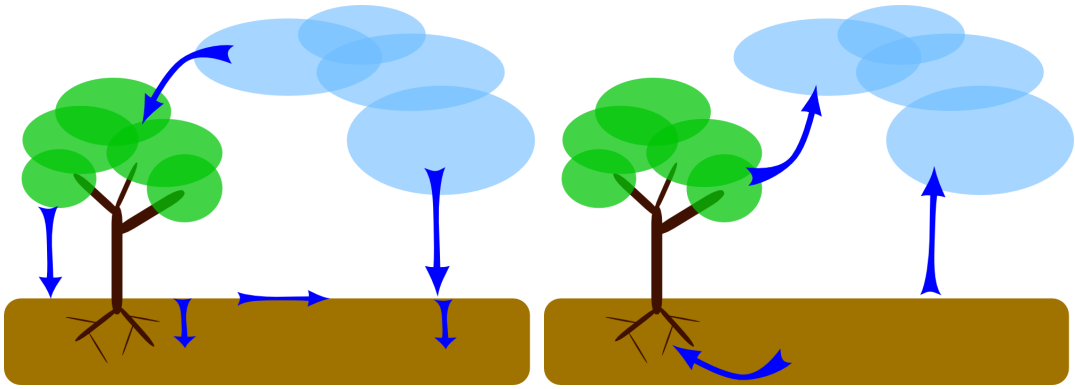

Fig. 2.1 : Schématisation du cycle de l'eau. À droite la partie « descendante » du cycle, avec les précipitations, le ruissellement et l'infiltration, à gauche la partie montante avec la prise d'eau, la transipration et l'évaporation.

Le pas de temps est journalier, un compromis entre processus rapides (ruissellement), processus lents (croissance des plantes) et données disponibles (en particulier climatiques).

Les précipitations sont simplement lues dans un fichier climatique, en même temps que la température. Ces données climatiques sont générées *a priori* à partir de données réelles mensuelles pour la région de Montpellier. Elles sont ensuite modulées par la topographie du terrain, en particulier un gradient vertical de température est établi.

Le ruissellement est considéré comme un processus diffusif suivant la surface du sol. On modélise explicitement une couche d'eau diffusant en fonction du gradient d'altitude. Le mouvement de l'eau dans le sol est aussi considéré comme diffusif sans phénomène d'advection, ce qui permet d'utiliser le même algorithme dans les deux cas. La différence tient au nombre de voisins considérés (8 en 2 dimensions pour le ruissellement, 26 en 3 dimensions pour l'eau dans le sol) et aux échelles de temps. Pour émuler la vitesse supérieure du ruissellement, on réalise un nombre paramétrable (plusieurs centaines dans les dernières simulations) de boucles de diffusion au lieu d'une seule pour l'eau dans le sol.

Le modèle de plantes utilisé est un GreenLab 1 très simplifié, avec des feuilles ne durant qu'un seul cycle, à expansion immédiate, et une structure à un seul axe de développement (modèle architectural de Corner). Ceci revient en fait à faire du modèle structure-fonction GreenLab un simple *process-based model*, décrit par une unique équation de récurrence, ce qui rend la simulation plus rapide et la paramétrisation moins lourde. Un des problèmes délicats à régler est la synchronisation du cycle de la plante (qui dépend de la température) avec la simulation journalière. L'algorithme divise le jour courant en deux parties dans le cas où on détecte la fin d'un cycle pour les plantes, et alloue la biomasse produite à chacun des deux cycles. On prend également soin de ne pas prendre plus d'eau dans le sol qu'il n'y en a... L'algorithme est décrit avec plus de détails dans l'article [Le Chevalier et al.(2007b)].

Enfin, l'évaporation du sol est décrite grâce à la formule de Turc [Turc(1961)]. On lui applique un facteur correctif si la zone est entièrement submergée, et le résultat est proportionné pour être cohérent avec le pas de temps (la formule s'applique normalement sur des périodes de 10 jours).

Logiciel

Le simulateur, appelé *LandVol*, a été implémenté en C++ avec un environnement de développement Qt sur un PC tournant sous Linux Fedora Core 4. L'aspect visualisation fait appel à la bibliothèque OpenGL. La figure 2.2 montre l'interface du logiciel. Cette interface permet à la fois de contrôler l'évolution de la simulation et de spécifier ou changer certains paramètres.

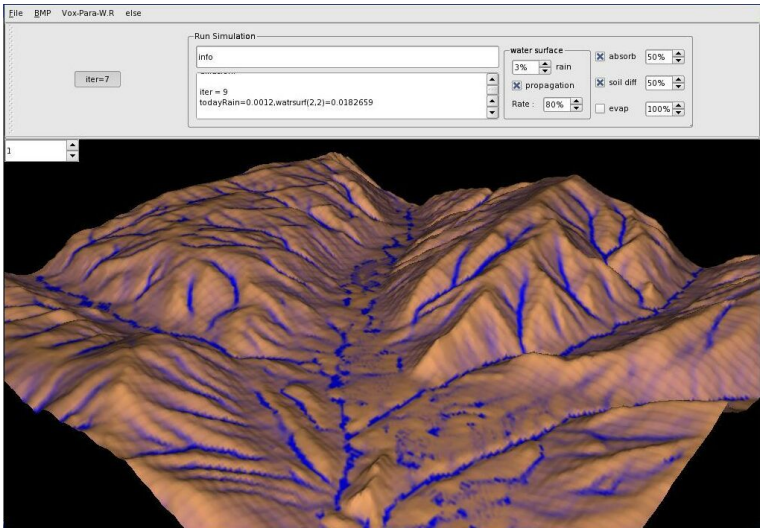

Fig. 2.2 : Interface du premier prototype de simulateur.

Comme la bibliothèque Qt permet d'utiliser des *threads* pour paralléliser les calculs, chacun des processus décrits plus haut est implémenté au sein de son propre thread, et les entrées-sorties et l'interface graphique ont aussi leurs propres threads. Ce prototype a ainsi été testé avec succès sur un calculateur parallèle, un BULL Novascale 5000.

Le composant intégré de visualisation affiche le maillage du terrain, avec une coloration dépendant de la quantité d'eau présente au point, et optionnellement de la biomasse. Il est également capable de représenter la biomasse sous la forme de « couronnes » (de simples sphères) de taille fonction de la valeur de la biomasse. La fonctionnalité d'export complet de l'espace voxel permet, pour un jour donné, d'utiliser les outils de visualisation volumique existants par ailleurs. Bien entendu ces exports complets occupent un espace disque significatif, et donc sont difficiles à utiliser pour étudier la dynamique jour par jour.

2.1.2 Problèmes

Bien que techniquement très satisfaisant, ce premier prototype présentait des faiblesses sur plusieurs plans.

D'abord les modèles utilisés sont trop simplistes ou simplement irréalistes. Seule une version très dégradée du modèle de plante est disponible, ce qui est dommage compte-tenu de la thématique de l'équipe et des dernières avancées dans ce domaine. Le modèle d'évaporation serait à vérifier mais il est probable que son utilisation au jour le jour ne soit pas validée. Enfin, le modèle de ruissellement est le plus gros point noir. L'algorithme de diffusion utilisé fait que le comportement de l'eau sur la surface s'apparente davantage au mouvement d'une pâte qu'à celui d'un fluide, ce qui pourrait être acceptable pour de la boue ou de la lave mais pas pour de l'eau... Il génère également des artefacts dus au sens de parcours des grilles, qui nuisent au caractère convaincant de la simulation. Il fallait donc rénover les modèles.

Ceci aurait pu être facilité si l'architecture logicielle avait commodément permis l'ajout ou la modification des modèles. Malheureusement, elle était très statique et ne pouvait pas accueillir les rénovations nécessaires. Le développement d'un second prototype a naturellement découlé de ces constatations.

2.2 Historique du second prototype

Le second prototype a été développé en partie au Liama durant mon stage de fin d'étude, et en partie à l'ECP durant le début de ma thèse, donc approximativement de l'été 2006 à l'été 2007, avec des petites modifications par la suite pour corriger des erreurs. Une description en a été publié dans [Le Chevalier et al.(2007a)].

La base de ce prototype était un module de calcul du ruissellement plus réaliste que celui du premier prototype. C'est un travail de modélisation et d'implémentation qui nous a éloigné des modèles de plantes et même de la plupart des aspects de modélisation intégrative, mais qui paraissait important pour atteindre des résultats plus crédibles.

Par la suite un module climatique et surtout un modèle de simulation de plantes ont été intégrés dans ce module. Nous avons fait ce choix car l'architecture logicielle du premier prototype n'était pas adaptée pour accueillir le nouveau simulateur de ruissellement.

En parallèle, un composant de visualisation fonctionnelle a été développé par Marc Jaeger, en se basant sur les structures de données et l'organisation du simulateur. Ce composant a grandement facilité le débogage du simulateur, nous permettant de vérifier en détail les comportements. Avec l'ajout de quelques post-traitements, on a pu vérifier la possibilité d'obtenir des images donnant une illusion de réalisme convaincante, comme présenté dans [Le Chevalier et al.(2007a)].

La suite du chapitre décrit certains points particuliers de ce prototype.

2.3 Ruissellement

Le ruissellement est une composante importante du cycle de l'eau, qui a aussi une influence sur les autres cycles, par exemple par le transport de matière en suspension.

Dans le cadre de notre étude, ce problème présente un certain nombre de particularités qui le rendent intéressant. Ces spécificités sont détaillées dans la section qui suit. La modélisation retenue, bien que simple, nécessite déjà une implémentation relativement complexe et coûteuse en terme de temps de calcul. Cependant, l'étude de ce problème apporte de nombreux éléments de réflexion au sujet des paysages fonctionnels, car ce genre de comportement se produit à chaque fois qu'il y a possibilité de saturation.

Le ruissellement a été beaucoup étudié, mais pas forcément d'une façon qui convienne à nos objectifs propres. Les hydrologues l'ont étudié, dans toute sa précision, pour déterminer les réponses à des évènements catastrophiques (pluies diluviennes, etc.) à une échelle de temps en particulier très inférieure à ce que l'on souhaite. Les géologues s'y sont également intéressés, bien sûr, mais avec le problème opposé, c'est-à-dire des hypothèses de temps et de stabilité des écoulements que l'on ne fait pas ici.

Finalement il est apparu que c'est le point de vue des géologues qui nous convient le mieux. Nous nous sommes basés principalement sur des travaux concernant la modélisation des réseaux de drainage sur des terrains numérisés [Tarboton(1997), Tucker et al.(1997)] et également sur quelques idées vues dans des travaux sur l'érosion, donc à une échelle de temps bien plus importante encore [Howard(1994)]. Il faut cependant noter que nous avons négligé tous les phénomènes d'érosion et de transport de sédiment, nous inspirant seulement des idées concernant le calcul des flux d'eau. Ainsi les altitudes sont considérées constantes dans la suite.

2.3.1 Spécificités du problème

Échelles

Le ruissellement se produit à une vitesse très différente du reste des phénomènes. Dans la mesure où pour le moment, on ne considère pas la circulation athmosphérique, le ruissellement est le phénomène le plus rapide dans notre paysage. Dans le cadre de ce prototype, on n'envisageait pas de simulation à un pas de temps autre que le jour. Il était donc nécessaire de simuler le ruissellement comme un phénomène global sur tout le paysage. En effet, en un jour, l'eau qui ruisselle a le temps de parcourir tout notre terrain, dont l'échelle spatiale est typiquement comprise entre 1 et 10 kilomètres.

Globalité

À cause de ces considérations d'échelle, le ruissellement ne peut pas se traiter localement. Bien sûr, il se modélise localement, mais la résolution doit être globale et chaque point peut avoir une influence sur tous les autres points. Il faut aussi noter que des phénomènes comme l'apparition de lacs donnent une complexité additionnelle :

non seulement le ruissellement en un point dépend de tous les points en amont, mais la topographie en aval est aussi importante puisque un lac peut se remplir et engloutir le point...

Le ruissellement peut être vu alors comme un opérateur complexe de redistribution de l'eau sur tout le terrain, avec une contrainte due à la limite d'absorption du sol. En entrée, on a les précipitations et le relief, en sortie, on a les teneurs en eau dans le sol, la quantité qui a ruisselé en tout point, et éventuellement les lacs qui se sont formés.

2.3.2 Modélisation

Modèle local

Localement, le modèle de ruissellement est relativement simple. Si l'on isole un point du paysage, on a :

- de l'eau qui entre sous forme de précipitation, P
- de l'eau qui arrive par ruissellement, R_i
- de l'eau qui est absorbée dans le sol, A
- de l'eau qui sort par ruissellement, R_o

On peut éventuellement rajouter un terme d'évaporation, E, qui ne change pas vraiment les algorithmes. Finalement le bilan local donne :

$$R_o = \max(0, P + R_i - E - A)$$

Ce modèle est schématisé sur la figure 2.3.

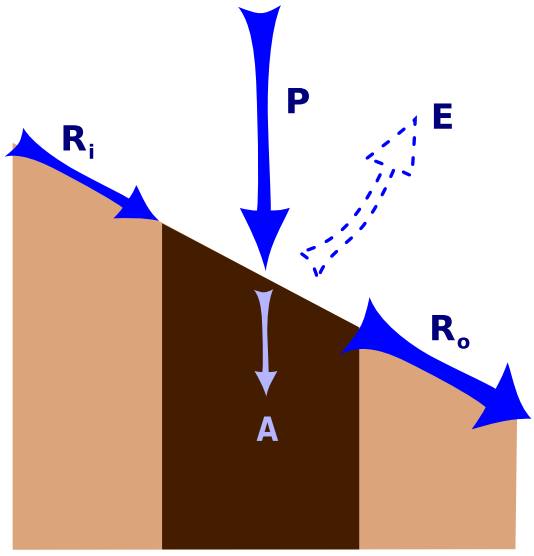

Fig. 2.3 : Schématisation du modèle local pour le ruissellement. On a représenté en pointillé l'évaporation, car le phénomène n'a pas été implémenté dans le prototype...

Il est possible que le point absorbe toute l'eau incidente ce qui donnerait un ruissellement nul en sortie. Le terme d'absorption A dépend de la teneur en eau et des

caractéristiques du sol sous-jacent. Il faut noter que l'on n'a pas besoin ici d'un modèle de sol complet. En fait, une grandeur synthétique, à savoir la quantité d'eau absorbable, suffit. En retour, le calcul de ruissellement fournit au modèle de sol une autre grandeur synthétique, la quantité d'eau absorbée.

Un facteur important est la répartition du ruissellement sortant vers les voisins immédiats du point. Va-t-on tout donner au plus bas voisin ? Répartir à proportion de la pente ? Chaque stratégie a ses avantages et ses inconvénients. Plusieurs sont exposées dans l'article [Tarboton(1997)] avec une analyse comparative. L'auteur s'intéresse ici au problème de la détermination de l'aire de drainage, qui est un peu moins général que ce que nous nous sommes fixé comme objectif pour le ruissellement, mais les stratégies de répartition demeurent valables. Là encore, l'algorithme ne dépend pas de la stratégie sélectionnée. Pour éviter les artefacts les plus évidents, nous avons choisi de répartir l'eau à proportion de la pente vers le voisin (une méthode dénotée MS dans l'article précédent). Ainsi le plus bas voisin est dominant, mais pas sur-favorisé. En fait cette méthode donne un caractère assez diffusif à l'écoulement, mais c'est la plus simple permettant d'éviter la discrétisation totale des directions d'écoulement.

Intégration spatiale

Considérant le bilan local, on voit que le terme R_i dépend des voisins du point. Il faut donc, avant de calculer le terme R_o et la quantité d'eau absorbée, avoir fait le calcul pour tous les voisins pouvant contribuer au ruissellement au point considéré. Et ainsi de suite... On peut définir ainsi un algorithme récursif, qui cherche à évaluer le ruissellement en remontant les pentes, jusqu'à arriver aux points qui n'ont pas de voisins plus haut qu'eux, pour lesquels le bilan est facile.

Les lacs

L'algorithme récursif ne règle pas tous les problèmes et en particulier ne prend pas automatiquement en compte les lacs.

En effet, qu'arrive-t-il cette fois aux points qui n'ont pas de voisins plus bas qu'eux vers lesquels le ruissellement peut s'écouler ? L'eau s'y accumule, bien sûr. Ces points particuliers sont désignés dans la suite sous le nom de *creux*. Le problème se pose lorsque cette accumulation d'eau provoque une hausse du niveau d'eau telle que le niveau devienne supérieur à l'altitude des voisins. Il faut alors distribuer une partie de cette eau aux voisins, jusqu'à ce que l'équilibre soit atteint, c'est-à-dire que l'ensemble des voisins du lac soient plus haut que le niveau d'eau, ou que le lac déborde. Un débordement se produit quand l'un des voisins du lac est plus bas que l'un des points du lac. De tels points sont appelés ici *sorties*. Un schéma de ces concepts peut être vu figure 2.4.

Les références que nous avions proposaient diverses solutions à ce problème : modification du terrain, recherche explicite des lacs... Mais toujours en faisant l'hypothèse que les lacs étaient des artefacts, ou qu'ils débordaient à coup sûr. Aucune de ces hypothèses ne convenaient, et il nous a fallu développer notre propre adaptation d'un

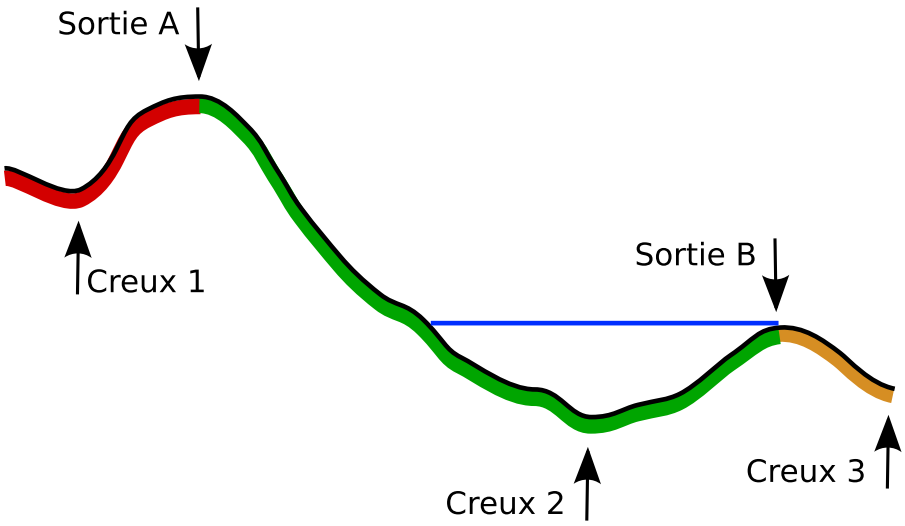

Fig. 2.4 : Concepts et principe de résolution du problème des lacs. On montre ici un terrain monodimensionnel présentant 3 creux. Un lac est représenté se formant autour du creux 2. Les points atteints par chacun des creux en remontant vers l'amont sont colorés de couleurs différentes : rouge pour le creux 1, vert pour le creux 2, orange pour le creux 3. Les intersections de ces zones permettent de repérer les points correspondants aux sorties. Le creux 1 a ainsi pour sortie la sortie A, les creux 2 et 3 ont pour sortie la sortie B. Les débordements séquentiels des lacs les uns dans les autres permet de régler ce problème de sorties identiques.

mécanisme de recherche. On voit qu'un algorithme de remplissage explicite du lac peut être très coûteux, parce qu'il impose de constituer une liste des points du lac, qu'il faudra recréer depuis le début à chaque fois que le calcul de ruissellement sera appelé. En effet, on ne peut pas facilement réutiliser les lacs des calculs précédents, parce que l'on ne maîtrise pas les changements de la capacité d'absorption et des précipitations. L'idée a donc été de rechercher un moyen d'éviter cette coûteuse extension pas à pas. Pour ce faire, on peut tirer parti d'arguments topologiques pour la recherche des sorties, et du parcours des points lors du calcul de ruissellement pour la recherche de l'équilibre.

Les points du terrain peuvent être répartis en plusieurs catégories en fonction de leur relation avec les creux. Certains ne sont atteints qu'à partir d'un creux du terrain par l'algorithme récursif. D'autres au contraire sont atteints par plusieurs creux. Cette différence peut être exploitée pour la recherche des sorties. En effet, la sortie d'un lac est nécessairement dans la seconde catégorie : il y a un chemin montant vers elle en partant du creux que l'on remplit, et un chemin descendant vers un autre creux où l'on suppose pour le moment qu'il n'y a pas de lac. C'est de plus le premier point de ce type qu'un algorithme remplissant le lac par ordre de hauteur atteindrait.

Finalement, pour un lac ayant pour origine un creux c donné, on voit ainsi que la sortie est le point le plus bas atteint par plusieurs creux et ayant un voisin atteint uniquement par c. Si le terrain était discrétisé avec une précision infinie et sans zone

plate, cette définition serait capable de déterminer la sortie de chaque lac en formation. Une autre conséquence est que le lac ne peut être constitué que de points qui ne sont atteints que par le creux associé, tout autre point se situant plus haut que la sortie. Sur le terrain discrétisé, on utilise un marqueur numérique pour identifier quel creux atteint une cellule donnée. La figure 2.4 montre un tel marquage pour le cas unidimensionnel, plus simple. La figure 2.5 montre le résultat de cet algorithme de marquage sur un terrain synthétique, une pente uniforme comportant cinq creux.

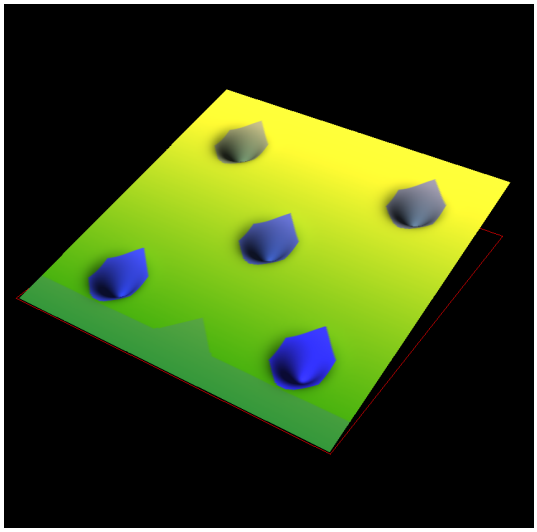

Fig. 2.5 : Marqueurs portés par les cellules du terrain dans un cas synthétique simple. Le terrain est coloré de vert à jaune en fonction de l'altitude. La nuance de bleu indique le marqueur. Toute l'eau s'écoulant de cellules portant la même nuance de bleu arrive dans le même creux. Certaines cellules n'ont pas de marqueur, cela signifie que l'eau qui en sort aboutit à de multiple creux. La sortie associée à un creux donné est la cellule la plus basse sur la bordure de la région marqué par l'identifiant du creux.

Du coup la réserve possible du lac peut être estimée pour chaque niveau. La réserve possible est la quantité d'eau qui est absorbée dans le lac, eau libre bien sûr, mais aussi eau absorbée par les points du sol inondés (qui du coup deviennent forcément saturés, même si le ruissellement seul n'y suffisait pas). Ce dont on a évidement besoin, c'est la valeur de cette réserve pour tout niveau d'eau, pour pouvoir ensuite calculer, en fonction de l'eau qui arrive au fond du creux, l'ensemble des points inondés simplement en fonction de leur altitude.

Ce calcul peut être effectué pendant l'algorithme récursif. À chaque fois qu'on atteint un nouveau point, on vérifie s'il est atteint par le creux de départ. Si c'est le cas, alors il est peut-être dans le lac qui va se former. On doit donc rajouter la quantité d'eau que ce point peut contenir à la réserve du lac pour tous les niveaux supérieurs à son altitude. D'un point de vue pratique, cela est possible en définissant pour chaque creux une liste de « remplissages » qui regroupe surface du lac, niveau d'eau, et réserve dans le lac, triée par niveau d'eau. Cette liste peut être mise à jour

itérativement au cours de l'algorithme récursif.

Au final, l'algorithme procède en trois phases principales. D'abord, une sortie est associée à chaque creux suivant le marquage du terrain. Ensuite, on effectue le calcul récursif du ruissellement, en créant au passage les listes de remplissage associées à chaque creux. Bien sûr on arrête de s'occuper de ces listes de remplissage au moment où on dépasse l'altitude de la sortie... Enfin, les débordements des lacs sont détectés. Lorsqu'un lac déborde, l'eau aboutissant dans le creux est reroutée vers la sortie associée, et on doit réitérer l'algorithme récursif de ruissellement. La boucle se poursuit jusqu'à ce que plus aucun lac ne déborde.

Le résultat de cet algorithme est représenté sur la figure 2.6. Le terrain synthétique utilisé est le même que sur la figure 2.5.

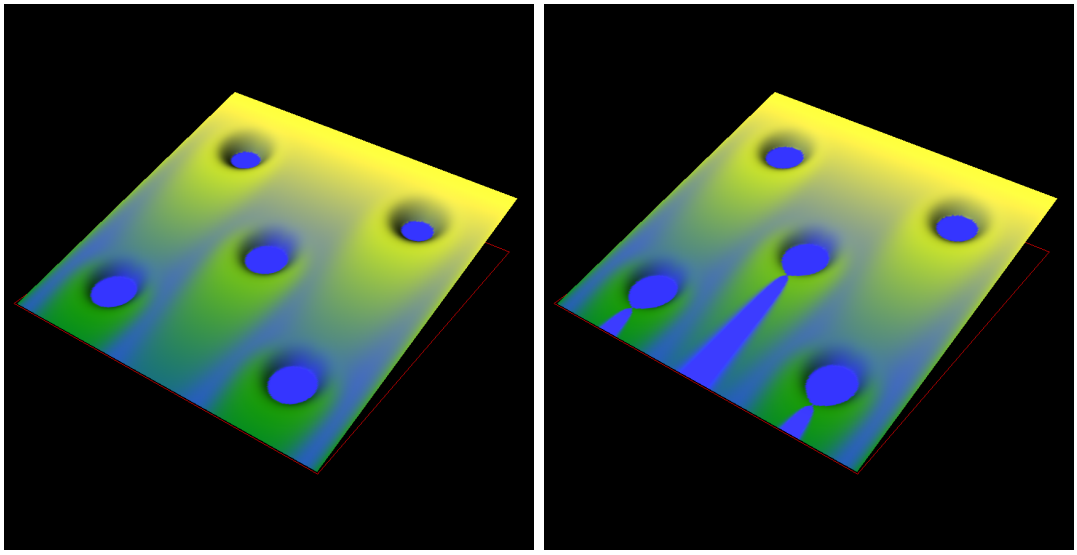

Fig. 2.6 : Deux simulations de ruissellement sur le terrain synthétique. La nuance bleue indique la quantité d'eau passant par le point. La valeur de la précipitation est plus importante sur l'image de droite, donnant lieu à un débordement des trois lacs inférieurs.

2.4 Implémentation et résultats

L'implémentation a été réalisée en C++ pour la partie simulation, en C et C++ pour la partie visualisation, avec utilisation des bibliothèques Glut et OpenGL. Le simulateur est un outil purement en ligne de commande, qui ne fait que produire des fichiers de données qu'on explore par la suite avec le visualiseur.

Le simulateur est divisé en plusieurs couches (*layers*), chacune représentant un processus ou objet différent. On a ainsi :

- `TerrainLayer` stockant les informations sur l'altitude et les paramètres du sol
- `WaterLayer` stockant les informations sur l'eau libre à la surface du terrain (ruissellement et lacs)

– `PlantLayer` stockant les informations sur les plantes
– `Climat` stockant les informations climatiques

Chacune de ces couches est en fait dérivée d'un même objet de base appelé `TimeStepper`. C'est la classe de base de tout objet pourvu d'une méthode `step()` qui représente la simulation sur un pas de temps. Un objet `TimeKeeper` se charge de la simulation proprement dite. Il conserve une liste de `TimeStepper`s dont il appelle successivement les méthodes `step()`, et incrémente le temps courant en fonction du pas de temps spécifié par l'utilisateur.

Chaque couche peut éventuellement envoyer des informations sur la console afin de donner des détails d'exécution. Un `TimeStepper` particulier se charge d'écrire les fichiers de sortie, en extrayant des informations provenant des différentes couches.

Cerrtaines couches sont structurées spatialement. `TerrainLayer` et `PlantLayer` sont ainsi dotées d'une spatialisation raster identique. `WaterLayer` se base également sur cette même spatialisation, mais en interne est plutôt stocké sous forme de graphe (on fusionne certains nœuds adjacents pour les besoins de l'algorithme de ruissellement). Des méthodes d'interpolation adaptées permettent de transmettre les informations d'une couche à l'autre lorsque leurs spatialisations diffèrent.

Le fichier de sortie est structuré sous la forme d'une grille dont chaque point représente une parcelle du paysage. Sur chacune de ces cellules, plusieurs données sont stockées (altitude, profondeur de l'eau, ruissellement passant au point, eau disponible pour les plantes, facteur environemental, production journalière de biomasse, production cumulée de biomasse, cycle courant du modèle de plante). À partir de ces informations et de la lecture du fichier de données climatiques (qu'on suppose uniformes sur le terrain), le visualiseur peut construire des représentation diverses, variant en particulier la couleur ou le mélange de couleur en fonction des valeurs de l'un ou l'autre champ, ajoutant des effets en fonction des précipitations... Il produit en outre des graphes de l'évolution temporelle des grandeurs moyennées sur le terrain.

Des résultats sont présentés plus en détail dans [Le Chevalier et al.(2007a)], ainsi que des détails sur la modélisation interne de chacune des couches. Nous nous sommes rendu compte que la grande difficulté est l'interprétation des résultats, dès lors que l'on se place dans des cas sensément plus réalistes où les variabilités spatiales et temporelles se multiplient. Ce problème est d'autant plus gênant que nous avions une grande incertitude sur les paramètres que nous utilisions, et étions dans l'incapacité de comparer nos simulations à des données réelles. La figure 2.7 rappelle des résultats et donne une idée de ce qu'il est possible d'obtenir avec ce prototype.

2.5 Problèmes

Malgré ses résultats prometteurs, un certain nombre de défauts rédhibitoires nous ont conduit à abandonner le développement de ce second prototype. Ils tiennent essentiellement en deux points détaillés ci-après, le partage des ressources et la synchronisation des processus.

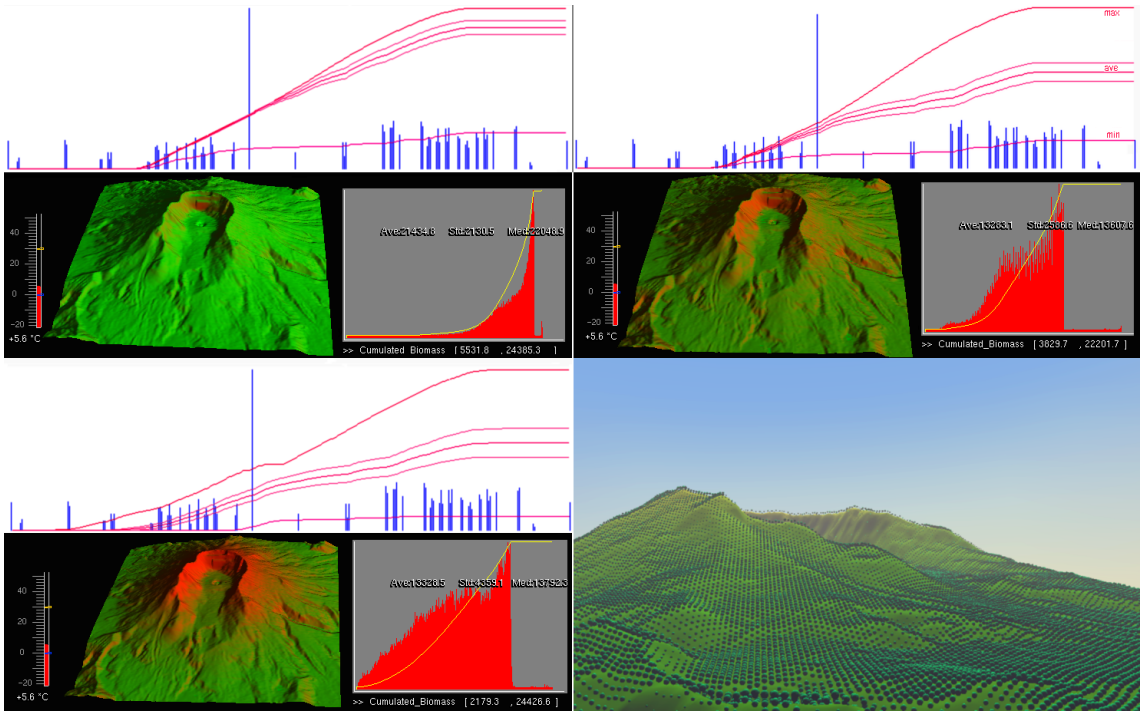

Fig. 2.7 : Simulation et visualisation avec le second prototype de paysage fonctionnel. De haut en bas, de gauche à droite : pas de limitation, limitation en eau, limitation en température et en eau, visualisation semi-réaliste avecgéométrie déduite de la biomasse et post-traitements. Les courbes représentent l'évolution statistique de la biomasse cumulée au fil du temps (on a une courbe pour la moyenne sur le terrain, une pour le minimum, une pour le maximum, et deux autres représentant l'écart-type). En dessous de chaque ensemble de courbe une vue 3D montre la variabilité spatiale et sa corrélation au terrain. Des histogrammes journaliers de biomasse cumulée sont aussi représentés, avec un arrière-plan gris.

2.5.1 Partage des ressources

La modélisation et la formalisation du comportement des ressources (et spécifiquement l'eau, dans le cas de ce prototype) se sont révélées insuffisantes. En explorant l'architecture logicielle précédement décrite il est facile de se rendre compte que les couches représentent en réalité à chaque fois des processus qui agissent sur les ressources, les transportant à travers le paysage ou les utilisant pour produire de la biomasse par exemple. À aucun moment n'apparaît une couche « ressource » qui aurait la charge de veiller à la bonne conservation de la ressource.

Au contraire, chaque processus stocke la valeur de la ressource sur laquelle il opère. Malheureusement, certaines de ces ressources « internes » sont en réalité des ressources partagées avec d'autres processus. Par exemple, l'eau contenue dans les couches superficielles du modèle de sol est également celle que les plantes utilisent pour leur croissance. Ce problème de partage est résumé sur la figure 2.8.

De ce fait, les processus doivent individuellement actualiser la valeur de leurs res-

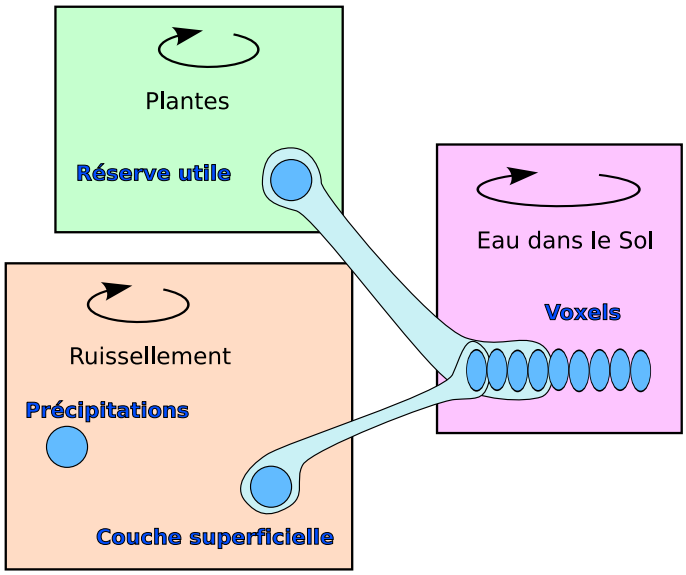

Fig. 2.8 : Les problèmes de partage de ressources dans le second prototype.

sources au début de chacun de leur calcul, car elle est susceptible d'avoir été modifiée par un autre processus. Ceci donne lieu à une quantité certes minime mais extrêmement gênante de code « parasite » qui ne sert qu'à synchroniser les ressources entre les processus. Ce code est dupliqué entre les processus mais leur impose de savoir avec qui ils partagent leurs ressources. De ce fait, la modularité de l'architecture est niée, car on ne peut échanger des processus librement. Le débogage du simulateur en est aussi considérablement complexifié, le test d'un élément indépendant étant rendu difficile par ce code parasite.

La compétition entre processus est aussi difficile à implémenter dans ce cadre. Étant donné que chaque processus possède ses ressources, il peut les utiliser sans aucune considération pour les autres. Il n'y a pas d'arbitrage entre processus partageant une ressource.

2.5.2 Synchronisation

Les problèmes du point de vue ressources masquent d'autres inconvénients plus fondamentaux de l'architecture de ce simulateur, qui concernent la gestion de la synchronisation temporelle des processus.

D'abord, le pas de temps est imposé de l'extérieur, sans aucune possibilité d'adaptation. Il peut varier, certes, mais jamais en fonction des résultats des calculs de chacune des couches. Ceci est principalement gênant pour ce qui concerne la saturation ou l'épuisement d'une ressource. Il n'y a pas de mécanisme intégré de détection de ce genre d'évènements.

Pire encore, l'évolution des processus n'est pas synchrone durant le pas de temps. Au contraire, chacun d'entre eux parcourt le pas de temps successivement, modifiant

ainsi les données pour tout ceux qui vont être exécutés ultérieurement. La simulation n'est donc pas cohérente, au sens où les calculs ne portent pas sur des données toutes valides à un même temps. Bien entendu, les implémentations des couches du simulateur tiraient parti de cette faille, et on peut même aller jusqu'à dire que certaines ne fonctionneraient pas dans un cadre imposant la synchronicité. En particulier, l'ordre dans lequel les processus sont traités est important et le simulateur ne fonctionne plus aussi bien si l'on inverse deux processus.

Les deux problèmes de la modélisation insuffisante des ressources et de la mauvaise gestion des phénomènes synchrones nous ont paru suffisament graves pour nous décider à arrêter complètement les développements sur ce prototype. Nous avons focalisé l'étude sur ces deux problèmes, d'abord sur la modélisation des ressources, puis à mesure que notre réflexion avançait sur ce point, sur l'étude de la synchronisation.

Les deux chapitres suivant décrivent les avancées effectuées sur chacun de ces deux axes. Nous exposons en premier le formalisme de synchronisation, car il donne un cadre conceptuel dans lequel les modèles de ressources et de compétition s'expriment naturellement. Mais il faut garder à l'esprit que les modèles d'interaction avec les ressources ont été développés et testés les premiers, et ont fortement influencé les idées de base du formalisme de synchronisation.

3. FORMALISME DE SYNCHRONISATION DE MODÈLES

Les prototypes et les réflexions sur la simulation de la compétition pour les ressources nous ont fait réaliser la nécessité de nous pencher sur une architecture de simulation générique capable de simuler des systèmes complexes formés par le couplage de modèles, et de synchroniser les calculs pour rester aussi cohérent que possible. Cette architecture et le formalisme de modélisation associé sont exposés dans ce chapitre.

3.1 Briques de base

Le formalisme de synchronisation utilise principalement deux sortes d'objets : les *Modèles* et les *Caches*. Les Modèles sont des composants de calcul, et les Caches sont des abstractions des données servant d'entrées et de sorties à ces calculs. Les Modèles lisent donc leurs entrées dans des Caches, puis écrivent leurs sorties dans d'autres Caches. Les Caches mettent alors à jour les données réelles de manière synchrone. Cette architecture basique est illustrée sur la figure 3.1. Nous utilisons dans toute la suite la convention d'écriture suivante : le terme Modèle employé avec une majuscule a le sens particulier qui est décrit dans ce chapitre, employé avec une minuscule il désigne le modèle au sens de représentation de la réalité, comme employé plus couramment par les scientifiques.

Une de ses propriétés les plus remarquable et utile est que les données de la simulation sont stockées dans une structure totalement séparée des composants de calcul, qui est toujours maintenue dans un état cohérent (en admettant que la mise à jour soit instantanée). Cette structure de données représente l'état de la simulation, et suffit à démarrer ou à reprendre une simulation. Non seulement cela permet des simulations plus stables, puisque les Modèles lisent toujours des données cohérentes, mais cela résoud aussi élégamment, en théorie, le problème de l'arrêt et et de la reprise des simulations, ainsi que celui de la sauvegarde indispensable des résultats.

Décrivons maintenant plus précisément les objets de base.

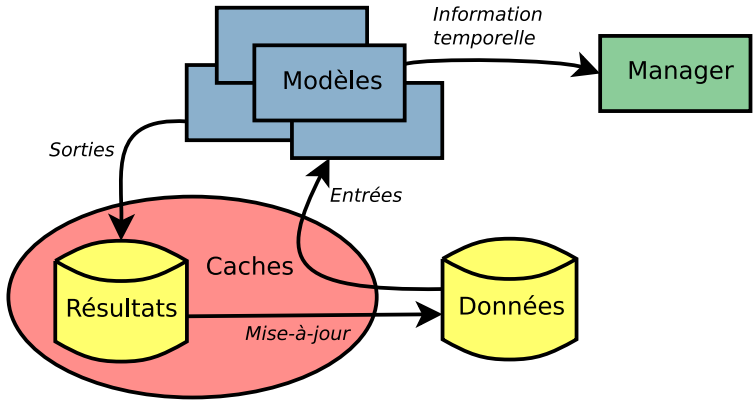

Fig. 3.1 : Schéma des objets de simulation et de leurs liens. On distingue principalement les Modèles, les Caches (il y a en fait plusieurs objets Caches non représentés ici), les données de la simulation et le Manager réalisant l'avancement dans la simulation. La structure des données de la simulation est dupliquée au sein des Caches, qui l'isolent des Modèles, ce qui assure une évolution synchrone. Les Modèles lisent leurs entrées dans les Caches, écrivent leurs sorties dans la structure dupliquée stockée dans les Caches. Ils transmettent aussi une information temporelle au Manager qui permet de fixer dynamiquement le pas de temps de la simulation. L'opération de mise à jour recopie les données locales des Caches vers les données de simulation. C'est cette opération qui est réalisée de manière synchrone.

3.1.1 Caches

Les Caches servent d'interface vers les données réelles, et font évoluer ces données réelles en les mettant à jour sur demande. Ils ont un lien vers les données réelles et une copie interne de ces données dans laquelle ils stockent temporairement les nouvelles valeurs, d'où la dénomination choisie.

Liens avec les Modèles

Les Caches communiquent avec les Modèles à travers deux interfaces spécifiques aux comportements différents, qui préservent la synchronisation des données de simulation. Ces deux interfaces permettent aux Modèles de collecter dans la liste des données externes les valeurs des variables d'entrée et de transmettre leurs variables de sortie.

La première interface est le *Getter*. Un Modèle peut, à partir d'un lien vers un Getter, lire une valeur. Dans ce cas le Cache renvoie la valeur stockée dans la structure de données externe. La seconde interface est le *Setter*. Un Modèle peut utiliser cette interface pour spécifier la future valeur à affecter à la donnée externe. Le Cache copie cette valeur et la conserve jusqu'à la prochaine mise à jour.

Comme le Getter est lié à la structure de données externes directement alors que

le Setter agit sur la copie interne située dans le Cache, il n'y a aucun risque que la valeur renvoyée par le Getter soit modifiée par un Modèle interagissant avec le Setter. En revanche, il est souhaitable qu'un seul Modèle puisse interagir avec un Setter donné, sans quoi la valeur en cache risque d'être écrasée. L'interface Getter n'impliquant aucune modification, plusieurs Modèles peuvent sans problème lire la valeur d'un même Getter.

Mise à jour

La valeur associée au Cache peut être mise à jour de plusieurs façons, suivant le type de la valeur et le système simulé.

La façon évidente de mettre à jour est de copier la nouvelle valeur dans la structure de données externe. Cependant, dans certains cas, conserver deux exemplaire d'une donnée peut être un gaspillage de mémoire. En particulier, quand la donnée contient une grande quantité d'information (une image par exemple) mais n'est modifiée que sur une petite partie de ces informations, une autre stratégie peut être choisie : on peut créer le Cache comme un Getter du « gros » type de donnée, et comme un Setter du type de donnée « variation », qui contient juste les informations nécessaires à la mise à jour.

En plus d'apporter des bénéfices en terme d'occupation mémoire et de vitesse d'exécution (car les opérations de copie sont d'autant plus coûteuses que les informations à copier sont nombreuses), ce choix peut être naturel pour certains modèles, qui par définition calculent la variation d'une donnée et pas une valeur.

Cohérence

Un Cache est dit dans un état cohérent lorsque la mise à jour ne causerait pas de changement significatif dans la valeur de la donnée externe. Significatif peut ici prendre plusieurs sens, selon les intentions et le choix de modélisation pour le couplage des Modèles. On peut être très strict et poser que tout changement de valeur est significatif, ou encore poser un seuil en deçà duquel on considère la variation comme négligeable. Cette définition est importante pour assurer la convergence du système dans certains cas, en particulier ceux comportant des boucles de rétroaction.

3.1.2 Modèles

Les Modèles sont responsables de tous les calculs de simulation. Ils peuvent être pourvus d'un état interne qu'ils doivent faire évoluer.

Entrées, Sorties, État

Un Modèle possède une liste d'entrées I, qui sont autant de liens vers des interfaces Getters.

L'état d'un Modèle est un ensemble de données qui lui sont propres, et ne sont accessibles en écriture que depuis l'intérieur du Modèle. Ces données sont accessibles par des Caches, comme les entrées et les sorties, sauf que dans le cas de l'état le Modèle possède le Cache et pas juste des liens vers des Getters et des Setters. Ainsi, on respecte le principe de séparation des données et des calculs, l'état des Modèles étant stocké dans la structure de données externe. Cela est nécessaire car un Modèle pourrait éventuellement ne pas être persistant en mémoire, alors que son état devrait l'être. Un exemple de ce type sera détaillé dans la partie 3.2.3.

Certains Modèles n'ont pas besoin d'état, et dans ce cas l'architecture de simulation présentée ne leur impose rien de particulier. La présence ou non d'un état pour un Modèle n'affecte en rien les fonctions de calculs présentées ci-après.

Calculs

Un Modèle doit définir trois fonctions qui seront utilisées dans l'algorithme de simulation.

La première est la fonction *cumul* :

$$O_C(t) = cumul(t, t_{prev}, I(t_{prev})) \tag{3.1}$$

où O_C est une liste des sorties « cumulatives », t est le temps courant, t_{prev} est la date du dernier appel à *cumul*, et I est la liste des entrées.

Cette fonction permet de représenter l'évolution du Modèle durant l'intervalle de temps $[t_{prev}, t]$, pendant lequel les entrées I sont supposées constantes. En pratique ce sont souvent des opérations d'intégration dans le temps, qui cumulent au fil du temps les effets des entrées, d'où le nom de la fonction et des sorties associées.

La seconde fonction est la fonction $trans$:

$$O_T(t) = trans(I(t)) \tag{3.2}$$

où O_T est une liste des sorties « transitoires ».

Cette fonction permet au Modèle de réagir instantanément aux changements de ses entrées. Un exemple de Modèle purement transitoire serait un additionneur qui se contente de faire la somme de ses entrées. Un tel Modèle n'a pas de mémoire du passé, pas de notion d'écoulement du temps.

Il existe bien sûr des cas de Modèles utilisant les deux fonctions, comme on le verra dans les exemples aux chapitres 4 et 5.

À part ces fonctions de calcul, les Modèles définissent une troisième fonction *time*, utilisée pour la synchronisation :

$$t_n(t) = time(t, I(t)) \tag{3.3}$$

Ici t_n est la date la plus lointaine à laquelle un appel à la fonction *cumul* du Modèle doit avoir lieu. Ceci permet en particulier, pour les modèles faisant une intégration temporelle, de fixer une borne supérieure au pas de temps. Mais les Modèles peuvent aussi s'en servir pour générer des évènements, par exemple.

Exemple simple

Nous allons donner ici un exemple simple de Modèle utilisant les trois fonctions vues précédement. Le but du Modèle est, à partir de deux entrées $A(t)$ et $B(t)$, de fournir la valeur $S = A \int_t B$ en sortie.

On voit qu'un changement de la valeur de A entraîne instantanément un changement de la valeur de S, ainsi la sortie est transitoire. Le modèle, pour générer cette sortie, a besoin de calculer et conserver l'intégrale $\int_{t_0}^{t} B$, qui va donc constituer l'état e du modèle.

Il n'y a plus qu'à écrire les trois fonctions. La fonction *cumul* approche l'intégrale et stocke le résultat dans l'état :

$$e(t) = s(t_{prev}) + (t - t_{prev})B(t_{prev}) \tag{3.4}$$

La fonction *trans* calcule simplement le produit pour obtenir la sortie :

$$S(t) = A(t)e(t) \tag{3.5}$$

Enfin, la fonction *time* peut, pour des impératifs de précision par exemple, imposer un pas de temps fini Δt :

$$t_n(t) = t + \Delta t \tag{3.6}$$

On peut même imposer une borne sur la variation de e, par exemple si elle ne doit pas dépasser la valeur K on peut choisir plutôt :

$$t_n(t) = t + K/B \tag{3.7}$$

Reste à voir comment l'appel des fonctions des Modèles et des Caches est organisé pour aboutir à une simulation du comportement du système. Ceci fait appel au concept de hiérarchie et est détaillé dans la section suivante.

3.2 Hiérarchies

La spécification des Modèles et des Caches donne la capacité de construire des Modèles hiérarchiques, c'est-à-dire contenant un graphe de Modèles et de Caches.

3.2.1 Hiérarchie générique

La structure d'un Modèle hiérarchique est assez simple à comprendre. Un exemple est détaillé sur la figure 3.2. On a besoin de définir des Getters et Setters particuliers, qu'on appelle des Buffers, pour transmettre les entrées du Modèle aux sous-Modèles et pour recopier les sorties des sous-Modèles dans celle du Modèle. Ces objets ne sont

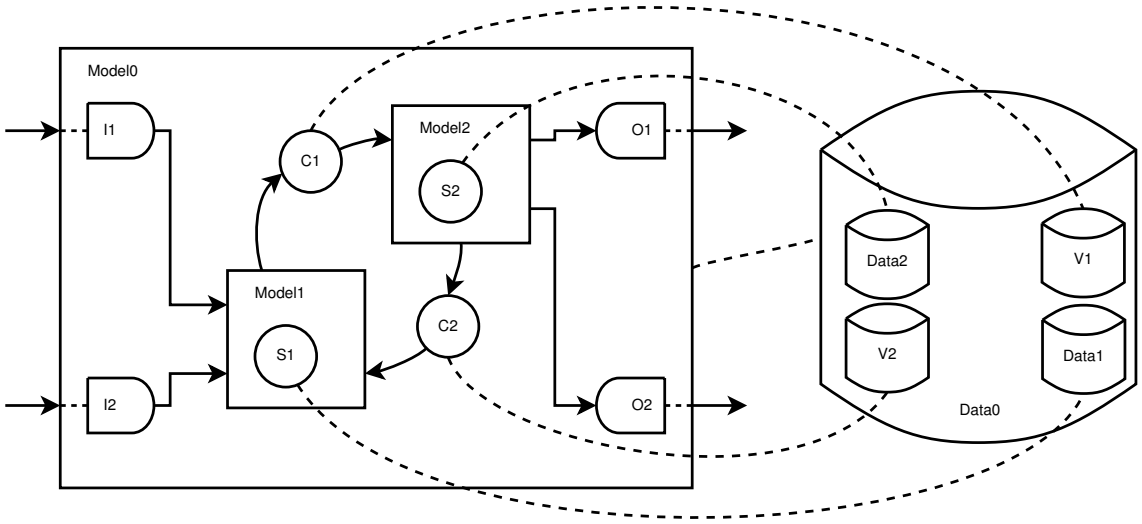

Fig. 3.2 : Représentation d'une hiérarchie de Modèles. Le Modèle Model0 contient deux sous-Modèles, Model1 et Model2, couplés par l'intermédiaire de deux Caches C1 et C2. Chacun des Modèles a aussi un état, S1 et S2 respectivement. Des buffers I1 et I2 permettent de lire les entrées, d'autres O1 et O2 permettent d'écrire des sorties. Des lignes pointillées relient les Caches vers les données de simulation, dont la structure est à l'image de celle du Modèle. Anisi, Data0 est la donnée d'état de Model0, et est composée de Data1 et Data2, les données d'état des sous-Modèles, et de V1 et V2, les valeurs associées aux Caches couplant les sous-Modèles.

pas des Caches, en ce sens qu'ils ne sont pas liés à une structure de données externe. Ils servent purement d'intermédiaires de calcul et n'ont pas à être synchronisés. C'est le Modèle qui est responsable de leur gestion et de leur utilisation.

Un Modèle hiérarchique a un état qui est représentatif de la structure interne, c'est-à-dire des Modèles et Caches qu'il possède. Il est entièrement responsable de la synchronisation de tous ces objets ; le Manager global de la simulation n'y a pas accès.

Le comportement d'un Modèle hiérarchique varie suivant la modélisation souhaitée. Il peut être un véritable sous-simulateur qui isole temporellement ses sous-Modèles du reste de la simulation (ce qui est une façon possible d'obtenir un effet de multi-échelle temporel). Il peut au contraire être simplement une « coquille » isolant des blocs logiques, mais n'ajoutant pas lui-même de comportements.

Deux exemples d'utilisation de hiérarchies sont détaillés dans la suite, pour le management de la simulation d'abord, puis pour certaines variantes de spatialisation.

3.2.2 Management de la simulation

Un manager de simulation est un type particulier de Modèle qui n'a pas d'entrées ni de sorties, et se contente de synchroniser les calculs de tous ses sous-Modèles. Il possède ainsi une liste M de sous-Modèles m_j, une liste C de Caches, et une représentation du graphe reliant les Modèles et les Caches.

La fonction *trans* du Manager est vide, et sa fonction *time* renvoie $+\infty$. Le reste de cette section décrit la fonction *cumul*.

La fonction *cumul* avance la simulation de t_0 à t_1 en plusieurs étapes, suivant les résultats des appels aux fonctions $time_j$ des sous-Modèles. Nous allons détailler ici cette fonction sous la forme d'un pseudocode, en la décomposant suivant les principes de la *programmation littéraire* (voir annexe A pour plus de détails à ce sujet, en particulier les références croisées générées sont expliquées au A.3). Le pseudocode complet d'un seul bloc est disponible en annexe B.

$\langle cumul(t_1, t_0)\rangle \equiv$ 1

$t_c \leftarrow t_0$
$t_p \leftarrow t_0$
faire
· $t_c \leftarrow \min(\{time_j(t_p, I_j)\}_j, t_1)$
· $\langle$*Avancée temporelle* 2$\rangle$
· $\langle$*Convergence* 5$\rangle$
· $t_p \leftarrow t_c$
tant que$(t_c < t_1)$

◇1 ;

Ici t_c est le temps courant de la simulation, et t_p est le temps précédent, utilisé quand la fonction $cumul_j$ est appelée pour chaque Modèle dans la partie *Avancée temporelle*. Il faut noter que contrairement à des approches précédentes pour les modèles de plantes décrites dans [Blaise et de Reffye(1994)], il n'y a pas ici d'échéancier qui fixe explicitement une séquence d'évènements futurs. Le système calcule la date de la prochaine évolution à partir de son état, et ne prévoit rien au-delà de cette date.

Dans le but d'optimiser légèrement les calculs, nous conservons deux listes A et U. A est la liste des Modèles affectés desquels les fonctions $trans_j$ ou $cumul_j$ devront être appelées, U est la liste des Caches affectés dans lesquels les Modèles ont stocké les résultats de leur calculs et qu'on doit peut-être mettre à jour.

On nettoie ces deux listes, puis on appelle $cumul_j$ pour les sous-Modèles :

$\langle$*Avancée temporelle*$\rangle \equiv$ 2

nettoyer A et U
pour tous les Modèles dans M
· $O_j^C \leftarrow cumul_j(t_c, t_p, I_j)$

◇2, 3 ; *1*

On remplit alors les listes en ajoutant les sorties cumulatives des Modèles dans U et les modèles eux-mêmes dans A, si leur état a changé. Finalement, une fois tous les calculs faits, on met à jour les Caches :

⟨*Avancée temporelle*⟩+ ≡ 3
· ajoute O_j^C à U
· **si** l'état du Modèle j a changé
· · l'ajouter directement à A
· **fin**
fin
⟨*Mise à jour des Caches* 4⟩
◇2, 3 ; *1*

Pendant la mise à jour des Caches ont lieu deux choses importantes. Premièrement, un Cache dans un état cohérent est complètement ignoré. Cela permet de converger vers un état stable (où tous les Caches sont cohérents) et optimise les calculs en supprimant des appels aux fonctions de calcul des Modèles affectés. Deuxièmement, les Modèles qui ont un Cache non cohérent comme entrée sont ajoutés à la liste A.

⟨*Mise à jour des Caches*⟩ ≡ 4
pour tous les Caches dans U
· **si** le Cache n'est pas cohérent
· · mettre à jour le Cache
· · ajouter les Modèles affectés par le Cache à A
· **fin**
fin
◇4 ; *3, 5*

La dernière étape est la convergence du système formé par les sous-Modèles et les Caches vers un état cohérent, grâce aux fonctions $trans_j$ des Modèles. Les fonctions $trans_j$ des Modèles de A sont appelées répétitivement jusqu'à ce que tous les Caches soient cohérents, ce qui fait que la liste A devient vide à l'itération suivante.

⟨*Convergence*⟩ ≡ 5
tant que A n'est pas vide
· nettoyer U
· **pour tous** les Modèles dans A
· · $O_j^T \leftarrow trans_j(I_j)$
· · ajouter O_j^T à U
· **fin**
· nettoyer A
· ⟨*Mise à jour des Caches* 4⟩
fin
◇5 ; *1*

La fonction $cumul(t_1, t_0)$ ainsi définie est générique et ne dépend pas du contenu des Modèles. Deux problèmes peuvent empêcher la simulation d'aboutir :
– le blocage temporel ($\exists j$ t.q. $time_j(t_p, I_j) = t_p$)
– la boucle infinie dans l'étape de convergence

Dans les deux cas, cela veut simplement dire que les modèles couplés sont incompatibles ou que le système évolue de manière explosive. Il faut alors regarder de près quel Modèle bloque ou comment évoluent les valeurs des Caches pendant la conver-

gence. Par expérience sur les premières applications, le premier cas est plus courant que le second ; il est finalement assez rare d'avoir un grand nombre d'itérations dans l'étape de convergence, puisque cela nécessite des rétroactions transitoires qui sont généralement modélisées indépendament comme la résolution d'une équation.

Il serait assez simple de généraliser les principes et d'obtenir un Manager pourvu d'entrée et de sorties, de façon à pouvoir construire une hiérarchie de Managers si besoin était.

3.2.3 Spatialisation

Les Modèles spatialisés peuvent aussi être décrits dans certains cas comme des Modèles hiérarchiques. Un Modèle spatialisé a des entrées et des sorties spatialisées. Un exemple typique et simple est une spatialisation sous forme raster, représentant une grille régulière de données homogènes sur un plan.

Il se peut que le comportement du Modèle spatialisé ne soit formé que par l'interaction de Modèles non-spatialisés appliqués sur chaque élément de la spatialisation, par exemple sur chaque cellule d'une grille raster. Il est alors logique de considérer le Modèle spatialisé comme contenant un grand nombre de sous-Modèles, peut-être interconnectés. La spatialisation vient d'informations spatialement hétérogènes stockées dans la grille, d'informations spatiales concernant la structure de grille elle-même, mais aussi de la topologie des liens établis entre les sous-Modèles. Le choix de la structure topologique des liens entre les sous-Modèles (par exemple le nombre de voisins considérés sur une grille, et leur pondération [Birch(2006)]) peut avoir un impact significatif sur les résultats de simulation, donc sur l'adéquation de la modélisation.

On a ainsi clairement une structure hiérarchique, venant d'une logique spatiale et plus temporelle comme dans le cas du Manager. Les deux pourraient cependant être associés.

L'utilisation de Modèles spatialisés a un inconvénient : comme les informations dans les Caches sont spatialisées, on ne peut plus optimiser en suivant quel Cache est mis à jour ou pas aussi efficacement. En particulier, il devient difficile de simuler de manière optimisée des phénomènes spatialement localisés ; on doit faire tourner le Modèle sur tout l'espace à la fois. Si ce cas de figure est courant avec les Modèles considérés, il vaut mieux alors ne conserver que des Modèles locaux connectés entre eux.

Cependant, l'établissement de telles connexions sur une spatialisation très détaillée (par exemple une grille avec de nombreuses cellules) peut notablement affecter les performances du système, en particulier en terme de mémoire. Si les évolutions se font généralement sur tout l'espace à la fois, l'utilisation de Modèles spatialisés hiérarchiques permet un autre type d'optimisation décrit ci-après.

La première constatation à faire est que tous les sous-Modèles sont identiques, à ceci près que leurs entrées, sorties et états pointent vers des données différentes. En général, les calculs de tous les sous-Modèles ne sont pas simultanés (il faudrait autant de processeurs qu'il y a d'éléments de spatialisation), il n'est donc pas utile de garder

en mémoire ni les sous-Modèles qui décrivent les mêmes fonctions, ni les liens entre ceux-ci et les données. On peut se contenter de quelques Modèles (un dans le cas mono-processeur) dont on change dynamiquement les liens pour leur faire parcourir tout l'espace au fil du calcul. Ceci impose un petit peu de travail supplémentaire au processeur, mais engendre un gain en mémoire significatif.

Cette méthode a été utilisée pour l'un des Modèles de l'application détaillée en 6.2. Bien sûr, en pratique ce n'est utile que lorsque le sous-Modèle a déjà été écrit et testé par ailleurs, sans quoi il peut se révéler plus simple de faire directement les calculs dans les fonctions du Modèle spatialisé sans passer par l'intermédiaire d'un sous-Modèle.

3.2.4 Possibilités de multi-échelle temporel

Le formalisme de modélisation exposé ici se prête naturellement à la réalisation de systèmes présentant une forme particulière de multi-échelle temporel. Il peut arriver, lors de la modélisation d'un système complexe, que l'on se retrouve dans la situation suivante : un des sous-systèmes A modélise une évolution à grande échelle temporelle, et est couplé avec un autre sous-système B qui modélise une petite échelle temporelle. Il se peut que l'on sache, ou que l'on veuille faire l'hypothèse, que les sorties de A ne sont vues par B que comme des contraintes constantes, et que réciproquement A ne voit de B que les sorties en régime permanent et pas le comportement transitoire.

On peut bien sûr modéliser ces systèmes sans aucune hiérarchie, mais dans ce cas les calculs de A vont être appelés très souvent, augmentant dramatiquement les temps de simulation pour un bénéfice douteux. Si effectivement le temps de réponse de A est très long par rapport aux évolutions de B, les étapes de calcul supplémentaires sont inutiles.

Dans ce cas là, il peut être utile d'encapsuler les Modèles de B dans un système qui réalise la simulation non dans sa fonction *cumul*, comme montré précédement pour les Managers, mais dans sa fonction *trans*, en se contentant d'aller chercher le régime permanent (ceci impose naturellement de définir les critères du régime permanent). Ainsi il ne fait que suivre les changements des entrées mais n'impose plus de pas de temps aux Modèles de A, et ne fait qu'exposer les valeurs de ses sorties après convergence, sans le détail des évolutions initiales.

Il va de soi que ce principe pourrait s'appliquer récursivement dans une véritable hiérarchie, permettant éventuellement de couvrir plusieurs ordre de grandeurs en échelle.

Cette réflexion souligne aussi une possible interprétation des fonctions des Modèles en terme d'échelle. Si on considère un Modèle à une échelle temporelle donnée, ou plus précisément à une granularité temporelle donnée, alors la fonction *cumul* regroupe tous les comportements dont la dynamique est décrite à cette granularité, et la fonction *trans* regroupe tous ceux se produisant à une granularité bien plus fine. Ces idées combinées à celles exposées précédement au 1.2.4 pourraient permettre de développer plus facilement des Modèles multi-échelle.

3.3 Description de l'implémentation

L'implémentation de l'architecture de simulation a été effectuée en C++ standard, avec utilisation de quelques conteneurs de la bibliothèque standard (en particulier les listes). L'usage de templates autorisé par le langage a permis de diminuer l'ampleur du code en réutilisant des comportements génériques, tout en permettant de gérer différents types de données avec une grande souplesse.

Deux hiérarchies de classes principales sont utilisées, l'une pour les Modèles, l'autre pour les Caches. Une fois les Modèles écrits, il ne reste qu'à les coupler dans un simulateur particulier (en fait un Manager). Dans l'architecture de simulation actuelle, le couplage est statique, c'est-à-dire que des simulateurs différents sont compilés pour chaque graphe de Modèles couplés, au lieu d'avoir une seule application dans laquelle des Modèles sont dynamiquement insérés.

3.3.1 Classes pour les Modèles et les Caches

La classe `Model` est une classe abstraite qui est la traduction directe du formalisme exposé plus haut, trois méthodes correspondant aux trois fonctions. Elle inclut simplement quelque fonctionnalités d'optimisation supplémentaires, qui peuvent dans certains cas éviter des calculs inutiles. De cette classe sont dérivés tous les autres Modèles, y compris le `Manager`.

Les Modèles dotés d'un état possèdent en plus une méthode `build(State* s)` qui permet de relier leurs Caches internes à une structure de données d'état externe. Ainsi on peut changer l'état d'un Modèle simplement en rappelant cette fonction.

Le Manager utilise des templates pour enrichir les Modèles et Caches utilisés, en ajoutant des informations relatives à leur parcours et à leurs relations, suivant cette technique :

```
struct ManagedModelBase: public virtual Model
{
    bool listed;
    PCacheList lcumul;
    PCacheList ltrans;
};

template <class M>
struct ManagedModel: public ManagedModelBase, M{};
```

Ainsi en utilisant la classe `ManagedModel<AModel>` on pourra utiliser le Modèle `AModel` sans que l'implémenteur de ce Modèle ait à se soucier des particularités du Manager. Le type `PCacheList` représente simplement ici une liste de ManagedCache, qui sont définis de manière analogue aux `ManagedModels`.

Ceci est une application de l'héritage multiple qui permettrait éventuellement de créer plusieurs classes `Manager` adoptant des stratégies de résolution et d'optimisation des calculs différentes, sans changer la classe `Model`.

La hiérarchie de classes pour les Caches fait aussi appel aux templates et à l'héritage multiple. Les classes de base `Updater`, `Getter` et `Setter` sont ainsi combinées pour créer les diverses variantes de Caches, les Buffers, etc. Tous les Caches sont des templates du type de donnée qui y est contenu, ce qui permet d'utiliser la puissance du système de gestion de types du C++ pour assurer la compatibilité entre les entrées et les sorties des Modèles, sans qu'il soit besoin d'aucun traitement supplémentaire. Cette souplesse peut cependant devenir un inconvénient car en donnant la pleine liberté aux modélisateurs sur leurs types en entrée et en sortie, on court le risque d'obtenir des Modèles incompatibles techniquement, alors qu'au prix d'un effort minime ils le seraient sur le fond (l'exemple typique serait l'utilisation par deux modélisateurs différents de deux types de gestion de tableaux différents).

3.3.2 Étapes pour le couplage de Modèles

Un couplage de Modèles est matérialisé par l'écriture d'une classe `Simulator` (bien sûr le nom est arbitraire) héritant de la classe `Manager`, contenant les Modèles à coupler. Les méthodes de simulation étant déjà intégrées dans la classe de base, il ne reste qu'à spécifier le système et les données d'état considérées.

La première étape est évidemment celle du choix des Modèles que l'on désire coupler. Il faut ensuite choisir la topologie de leurs relations, donc les Caches qui les relient les uns aux autres. À partir de ces listes de Modèles et de Caches on peut spécifier l'état du `Simulator` dans une structure `Simulator::State`, regroupant toutes les données nécessaires à la simulation. C'est souvent une image quasiment triviale des listes de Caches et de Modèles...

C'est le constructeur du `Simulator` qui va établir les liens entre Caches et Modèles, créant ainsi le graphe de couplage désiré. La méthode `build(State* s)` du `Simulator` va relier les Caches et les Modèles aux données d'état, appelant éventuellement les méthodes `build()` des sous-Modèles récursivement.

La fonction `main()` (ou équivalent, suivant l'interface utilisateur que l'on désirerait mettre en place) doit se dérouler en plusieurs étapes :

1. Création du `Simulator`
2. Création de la structure `State` contenant les données d'état
3. Initialisation des données d'état
4. Établissement du lien entre `Simulator` et données d'état (appel de la méthode `build()`)
5. Choix de l'intervalle de temps
6. Simulation
7. Écriture des résultats, par appel de fonctions de sorties sur la structure de données d'état

Quand on veut faire des sorties à plusieurs temps dans la simulation, on peut réaliser une boucle sur les trois dernières étapes, ou même écrire des Modèles spécialisés

dans l'écriture de résultats. La première solution est plus économe, la deuxième plus souple.

La classe `Manager` contient aussi une fonction permettant d'écrire sur le disque un graphe du système tel qu'il a été décrit. Ce fichier est dans le langage DOT, visualisable grâce aux outils de la suite GraphViz [Gansner et North(1999)]. Ce fichier est en théorie redondant mais extrêmement utile pour le débogage, étant donné que l'établissement des liens en C++ est un processus où des erreurs sont facilement commises, donnant des simulations complètement fantaisistes. Une visualisation graphique est bien plus parlante et montre beaucoup plus clairement les erreurs. Idéalement, il faudrait faire l'inverse, c'est-à-dire générer le C++ à partir de la description DOT, mais le temps a manqué...

3.4 Comparaison avec DEVS

Le formalisme DEVS (Discrete EVents Specification) est une approche existante permettant de construire et simuler des systèmes composés récursivement de sous-modèles [Zeigler et Vahie(1993), Zeigler et al.(2000)]. Il a été utilisé dans le contexte des simulations de paysages, en conjonction avec d'autres formalismes descriptifs comme décrit dans [Mayer et Sarjoughian(2007)].

Les modèles du formalisme DEVS sont décrits comme suit :
- un ensemble de valeurs en entrée X
- un ensemble de valeurs en sortie Y
- un état S
- plusieurs fonctions :
  - *transition interne* δ_{int}
  - *transition externe* δ_{ext}
  - *transition confluente* δ_{con}
  - calcul des sorties λ
  - avancement temporel t_a

Pour résumer, la fonction t_a renvoie la date à laquelle la fonction δ_{int} doit être appelée en fonction de l'état. Ce mécanisme est tout à fait similaire à notre fonction *time* et nous nous en sommes d'ailleurs inspirés.

La fonction δ_{ext} permet de réagir aux entrées, et donc tient un rôle analogue à notre fonction *trans*. En revanche, elle ne modifie pas les sorties mais uniquement l'état du modèle. Cette dernière particularité est aussi partagée par la fonction δ_{int}. La fonction δ_{con} ne sert qu'à indiquer l'ordre d'appel au cas où des évènements internes et externes arrivent simultanément (le choix par défaut étant d'appeler δ_{int} puis δ_{ext}).

La fonction λ est appelée pour calculer les sorties en fonction de l'état uniquement, et pas des entrées. Cet appel a lieu au moment où l'état change, sur la valeur précédente, et non sur la nouvelle valeur. Ce comportement n'est pas forcément intuitif dans tous les contextes et doit donc être gardé en tête...

Le formalisme DEVS originel et ses dérivés ont été implémentés dans plusieurs langages, donnant diverses possibilités techniques suivant l'ouverture du code source

et les choix de conception[1]. Une comparaison rigoureuse est extrêmement difficile et coûteuse en terme de développement, comme énoncé dans [Quesnel et al.(2009)], ce qui fait que paradoxalement la variété d'outils peut nuire à une adoption plus large de ce formalisme. Il faudra peut-être encore quelques années avant qu'une implémentation open-source incontournable n'émerge, même si VLE [Quesnel et al.(2009)] commence à endosser ce rôle dans les communautés avec lesquelles j'ai été en contact durant ma thèse (en particulier cette implémentation a été choisie comme base de la plate-forme RECORD [Bergez et al.(2009)] de l'INRA). Il peut paraître surprenant, compte tenu des capacités formelles démontrées par DEVS (en particulier ses capacités à s'adapter à divers formalismes allant des équations diférentielles aux systèmes multi-agents), que son utilisation ne soit pas plus répandue et qu'il ne soit pas encore plus réputé. Au minimum, on pourrait s'attendre à ce que toutes les équipes travaillant sur les évènements discrets l'utilisent, or ce n'est pas le cas... Dans le contexte des paysages, plusieurs formalismes ou plate-formes de développement sont disponibles et encore en évolution (par exemple celle décrite dans [Costanza et Voinov(2004)]).

Il est certain que le formalisme DEVS permettrait de développer des modèles et systèmes qui donneraient au final exactement le même résultat que notre architecture de simulation. Par ailleurs, nous n'avons pas la prétention d'être entièrement équivalents à DEVS (ce qui paraît difficile vu que les Modèles ont moins de fonctions). Notre choix de développer une architecture et un formalisme indépendant résulte d'autres facteurs.

Le premier est qu'en redéveloppant un nouveau formalisme, on contrôle complètement son implémentation. Ainsi nous sommes certains du fonctionnement interne jusque dans les moindres détails ce qui n'était le cas pour aucune implémentation de DEVS. La ré-implémentation nous permettait de comprendre la nécessité de chaque fonction et ainsi de nous assurer que nous ne rajoutions pas de complexité inutile pour nos objectifs. Ceci était particulièrement important dans l'optique du développement de Modèles spatialisés.

Ceci étant dit, nous aurions aussi bien pu créer une nouvelle implémentation du formalisme DEVS. Nous ne l'avons pas fait pour deux raisons.

Nous voulions formaliser la structure des données de simulation, et les séparer clairement des calculs effectués par les Modèles. Le formalisme DEVS ne détaille pas vraiment l'organisation des données, donc il aurait été nécessaire de compléter le formalisme en ce qui concerne l'interaction avec les données. À bien des égards on pourrait dire que notre formalisme est fondé initialement sur les données, ce qui le rend aussi plus directement adapté à l'arrêt, au redémarrage et à la sauvegarde de la simulation.

L'autre raison est que le formalisme DEVS, pour les modèles, est finalement assez mal adapté à ce que l'on voulait spécifier. Je pense que la cause de cette particularité est la source historique de DEVS (à savoir les systèmes à évènements discrets). Le formalisme est bien adapté à ces systèmes, mais moins à notre vision de la décomposition en sous-modèles du paysage. En particulier, nous avons peu de modèles à retards, pour

[1] voir une liste sur la page `http://www.sce.carleton.ca/faculty/wainer/standard/tools.htm`

lesquels les fonctions de DEVS facilitent l'implémentation. Au contraire, nous avons des modèles de calculs instantanés à partir des entrées, qui nécessitent un état un peu complexe à mettre en place pour être fidèlement représenté par un modèle atomique DEVS, même si cela est possible. Ceci découle en particulier de certains problèmes d'échelles temporelles. Comme on l'a vu au 3.2.4 le formalisme proposé ici possède une interprétation possible en terme d'échelle, absente dans DEVS. Notre formalisme est ainsi plus précisément adapté à nos besoins, dans le sens où l'implémentation des Modèles est très proche du concept que l'on en a. On peut aussi supposer, même si ce souci est secondaire à ce stade de notre travail, que l'on gagne en performance en choisissant d'implémenter un formalisme adapté au mieux aux modèles que l'on veut simuler.

3.5 Contraintes imposées aux modèles

Le formalisme de simulation impose peu de contraintes aux modèles, cependant une propriété au moins doit être assurée : le modèle doit être capable d'évoluer raisonnablement avec un pas de temps aussi petit que l'on veut.

Cette contrainte très simple peut avoir de grandes implications, en particulier le côté « raisonnable ». De nombreux modèles, particulièrement en biologie, ont un pas de temps implicite qui doit être respecté pour avoir une évolution cohérente. Certains modèles dit « discrets » (donc sans pas de temps explicite) sont en réalité des modèles ayant une part de continu qui doit être explicitée si on désire un couplage réaliste.

Un exemple, critique pour nous, de cette problématique, est le modèle de plante GreenLab. Le, ou plutôt les modèles GreenLab classiques (il en existe plusieurs versions) sont décrits comme discrets, mais les cycles représentés dans ces modèles ont en réalité une durée qui dépend de l'environnement. L'état de la plante à un cycle donné ne dépend que des cycles précédents. Mais en réalité, certains processus sont continus (par exemple la photosynthèse) et l'accumulation sur tout un cycle peut être absurde d'un point de vue biologique. Par exemple, l'idée de pool commun matérialisé se remplissant de biomasse au fil du cycle est adéquate quand on fait toujours les bilans sur un cycle entier, mais plus quand on fait de même sur des fractions de cycle : où se trouve ce pool commun dans la plante ? Nous avons été ainsi forcés de repenser totalement le modèle GreenLab dans l'optique de l'insertion dans l'architecture de simulation, comme présenté dans le chapitre 5.

Cette contrainte peut sembler excessive, et même préjudiciable à l'utilisation du formalisme et de l'architecture de couplage. Je la considère au contraire comme une opportunité de clarifier les modèles et leurs hypothèses cachées. Rien n'empêche d'insérer un modèle tel quel de toute façon, simplement le comportement ne sera peut-être pas satisfaisant. Dans le cas du modèle GreenLab, la formulation continue a ouvert des perspectives et souligné des possibilités nouvelles d'exploitation du modèle. Comme il sera rappelé plus loin en conclusion, c'est un des plus grands bénéfices de notre réflexion sur l'extension au niveau du paysage.

4. MODÈLES POUR LES RESSOURCES

Au sein des peuplements hétérogènes de plantes, notre cadre d'étude, l'un des phénomènes clés que nous souhaitons simuler est la compétition pour des ressources. Un des problèmes du second prototype était la formalisation très insuffisante de la compétition. Il a été nécessaire de conduire une réflexion plus poussée sur ce point, dont les résultats sont exposés dans la première section de ce chapitre. Dans la seconde, on trouvera un modèle très simplifié du comportement de l'eau dans le sol.

Les ressources sont définies ici comme des grandeurs physiques, ordinairement conservatives, qui sont échangées, transportées ou consommées par les processus du paysage. Dans un contexte agronomique, certains intrants peuvent être modélisés ainsi : eau, mais aussi nutriments, carbone...

4.1 Ressources et compétition

Nous avons dû spécifier des formalismes génériques d'interaction avec les ressources, et définir des Modèles qui les fournissent ou qui agissent sur elles, afin de pouvoir tirer parti de l'architecture de simulation qui avait été développée.

L'architecture de simulation est basée sur des flux d'information reliant les entrées et les sorties des Modèles. Mais ces informations ne sont pas des ressources, étant donné que la lecture d'une information ne la consomme pas, et qu'elle peut être copiée et partagée par autant de Modèles que l'on veut. L'interaction avec une ressource doit donc reposer sur l'utilisation de plusieurs flux d'information, que nous allons décrire dans ce qui suit.

4.1.1 Modèles d'interaction avec les ressources

L'interaction avec les ressources est modélisée par trois informations complémentaires utilisées comme entrées ou comme sorties selon que le Modèle met à disposition la ressource ou agit sur la ressource.

Un Modèle qui permet les actions sur une ressource a deux sorties : la quantité de ressource disponible pour le prélèvement (A) et la quantité de ressource qui peut être ajoutée avant saturation (F). Dans le cas très simple d'un seau d'eau, A serait la quantité d'eau dans le seau, et F la quantité d'eau qu'il faudrait ajouter pour remplir

le seau. Il a une seule entrée, le flux de ressource D sortant du Modèle (un flux entrant est bien sûr représentable par $D < 0$). Ce motif d'entrées-sorties est appelé un *Fournisseur*.

Réciproquement, un Modèle qui agit sur une ressource a deux entrées A et F, et une sortie D. Ce motif d'entrées-sorties est appelé un *Accesseur*.

Bien sûr, un même Modèle pourrait interagir avec plusieurs ressources à travers plusieurs Accesseurs et Fournisseurs. Plusieurs exemples sont détaillés dans le reste de cette section.

4.1.2 Stockage de la ressource : les Conteneurs

Les Conteneurs sont des Fournisseurs élémentaires qui stockent la ressource. Ils sont supposés indépendants des processus agissant sur les ressources. Leurs limitations sont la cause principale de la compétition. Plusieurs processus peuvent entrer en compétition sur un même Conteneur grâce à des Modèles appelés Répartisseurs, détaillés plus bas.

Le seau d'eau précédemment cité est un bon exemple de Conteneur. Le concept est illustré figure 4.1.

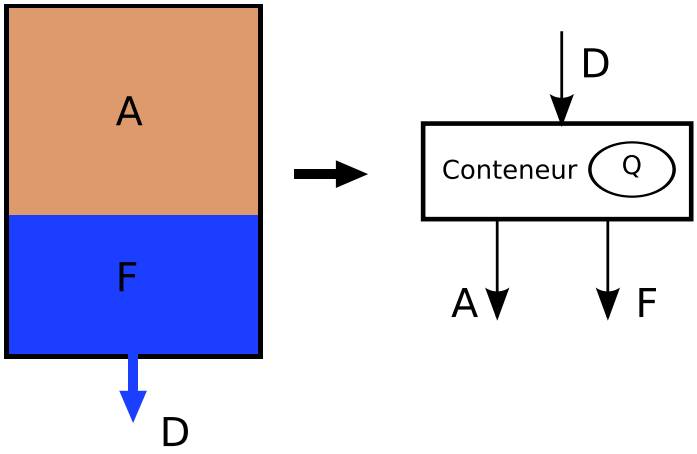

Fig. 4.1 : Illustration d'un Conteneur. À gauche, le cadre noir représente les limites de la ressource, la partie bleue est remplie par la ressource, la partie brune est vide. La demande D représente le flux sortant de ressource. À droite, un schéma du Modèle correspondant, avec son Cache interne pour l'état Q, ses deux sorties A et F et son entrée D.

L'état d'un Conteneur comprend la quantité de ressource actuellement stockée Q, ainsi que les quantités minimales Q_{min} et maximales Q_{max} qui peuvent être stockées. Les Conteneurs sont des Fournisseurs et à ce titre, ont deux sorties transitoires A et F, et une entrée D.

La fonction *cumul* met simplement l'état à jour suivant la demande imposée durant l'intervalle de temps :

$$Q = Q_{prev} - D(t - t_{prev}) \tag{4.1}$$

La fonction *trans* calcule A et F suivant l'état :

$$A = Q - Q_{min} \tag{4.2}$$
$$F = Q_{max} - Q \tag{4.3}$$

Et enfin la fonction *time* renvoie le temps avant que l'une des deux limites Q_{min} ou Q_{max} ne soit atteinte avec le flux courant :

$$t_n = \begin{cases} t_{prev} + \frac{Q - Q_{min}}{D} & \text{si} \quad D > 0 \\ t_{prev} + \frac{Q - Q_{max}}{D} & \text{si} \quad D < 0 \\ \infty & \text{si} \quad D = 0 \end{cases} \tag{4.4}$$

4.1.3 Compétition pour la ressource : les Répartisseurs

Les Répartisseurs sont les composants qui permettent à plusieurs objets d'interagir avec plusieurs Conteneurs. Ils contiennent des Fournisseurs qui agissent comme des sortes de Conteneurs virtuels, et des Accesseurs qui sont reliés à d'autres Conteneurs.

Ce sont des composants de communication, qui se contentent de transmettre et calculer des informations. Ainsi ils ne consomment pas de ressources et ne font qu'en diriger les flux. On pourrait imaginer plusieurs façons de construire cette communication, un modèle pondéré simple est présenté dans le reste de cette section.

Un Répartisseur n'a pas d'état. C'est un modèle transitoire pur, qui n'a pas d'évolutions spontanées. On a donc $t_n(t_{prev}) = \infty$ et la fonction *cumul* ne fait rien. Toutes les sorties sont transitoires.

Nous allons commencer par les Répartisseurs les plus simples, ceux avec un seul Accesseur (Partageurs) et ceux avec un seul Fournisseur (Groupeurs), puis les utiliser pour construire un Répartisseur générique avec plusieurs Accesseurs et Fournisseurs.

Partageurs

Les Partageurs sont utilisés pour partager une ressource entre plusieurs Modèles.

Un Partageur a N Fournisseurs $\{(D_i, F_i, A_i)\}_{i \in \{1,\dots,N\}}$, et un Accesseur vers la ressource à partager (D, F, A). La force du lien entre l'Accesseur et le Fournisseur i est représenté par un coefficient α_i.

Les $2N+2$ entrées sont donc α_i, A, F, D_i. Les $2N+1$ sorties sont A_i, F_i, et D. La fonction transitoire calcule les sorties en fonction des entrées. Ce type de Répartisseur est représenté sur la figure 4.2.

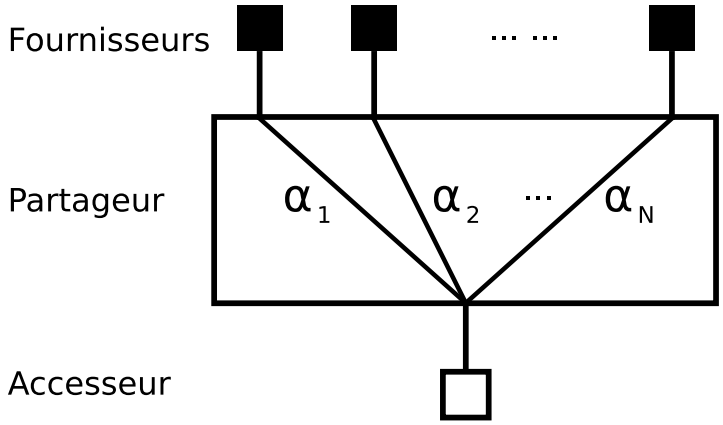

Fig. 4.2 : Illustration conceptuelle d'un Partageur. Les N Fournisseurs sont représentés par des carrés pleins, l'unique Accesseur par un carré creux, et les lignes à l'intérieur représentent les liaisons établies, avec les poids α_i associés.

Comme le Partageur ne peut absorber de ressource, ceci impose que la demande soit conservée. Ainsi le calcul de D est facile :

$$D = \sum_{i \in \{1,\dots,N\}} D_i \tag{4.5}$$

Le reste des sorties est simplement calculé avec une relation linéaire en utilisant les coefficients α_i, $i \in \{1, \dots, N\}$:

$$F_i = \alpha_i F \tag{4.6}$$

$$A_i = \alpha_i A \tag{4.7}$$

Groupeurs

Les Groupeurs sont utilisés pour regrouper plusieurs ressources (de même nature, de préférence) et faire en sorte qu'elles soient vues comme une seule par un processus.

Un Groupeur a M Accesseurs $\{(D_j, F_j, A_j)\}_{j \in \{1,\dots,M\}}$, et un Fournisseur (D, F, A). La force du lien entre le Fournisseur et l'Accesseur j est représentée par un coefficient β_j. Ce type de Répartisseur est représenté sur la figure 4.3.

On a donc $3M + 1$ entrées β_j, A_j, F_j, D. Les $M + 2$ sorties sont A, F, et D_j.

F et A sont simplement calculés par la relation linéaire :

$$F = \sum_{j \in \{1,\dots,M\}} \beta_j F_j \tag{4.8}$$

$$A = \sum_{j \in \{1,\dots,M\}} \beta_j A_j \tag{4.9}$$

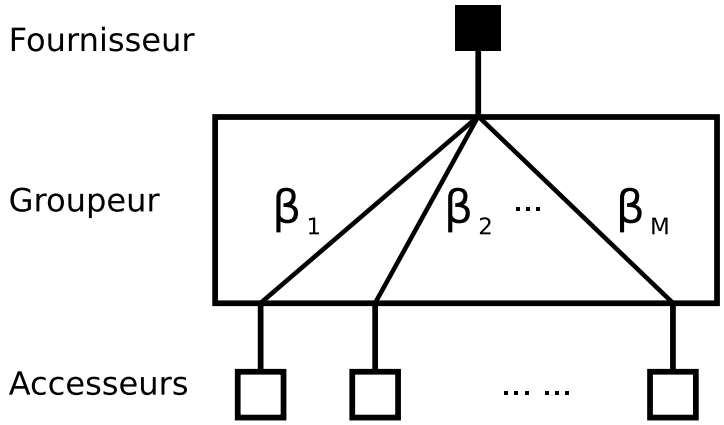

Fig. 4.3 : Illustration d'un Groupeur, suivant les mêmes conventions de représentation que sur la figure 4.2, avec cette fois les β_i comme poids associés.

Le calcul des demandes D_j est un peu plus complexe. La première contrainte que l'on veut vérifier est $\sum_j D_j = D$, en raison de l'hypothèse générale de conservation des ressources au travers des Répartisseurs. Mais ceci laisse encore beaucoup de possibilités... Notre objectif a été de répartir la demande aussi équitablement que possible entre les Accesseurs, suivant l'état de chacune des ressources. Ceci est fait de façon à maximiser le temps de fonctionnement possible (fonction *time*) des Conteneurs.

Dans ce but on définit plusieurs quantités suivant le signe de D :

$$C = \begin{cases} A & \text{si} \quad D > 0 \\ -F & \text{si} \quad D \leq 0 \end{cases} \tag{4.10}$$

$$C_j = \begin{cases} \beta_j A_j & \text{si} \quad D > 0 \\ -\beta_j F_j & \text{si} \quad D \leq 0 \end{cases} \tag{4.11}$$

$$\tau = \frac{C}{D} \tag{4.12}$$

$$\tau_j = \frac{C_j}{D_j} \tag{4.13}$$

Ici τ est l'intervalle de temps avant la saturation ou l'épuisement de la ressource représentée par le Fournisseur, et τ_j est l'intervalle de temps avant l'épuisement ou la saturation de la ressource vue par l'Accesseur j. C et C_j sont les disponibilités en ressources correspondantes, différentes suivant que la ressource est en cours d'épuisement ou de remplissage. Les signes de F et F_j sont corrigés de sorte que τ et τ_j soient toujours positifs. L'apparition du temps dans ces équations est aisée à comprendre en faisant le lien avec l'équation (4.1), elle résulte du rapport entre une quantité (disponibilité de la ressource) et un flux de cette même quantité (demande).

On calcule les demandes D_j de sorte que tous ces temps de fonctionnement soient égaux :

$$\tau_j = \tau, \ \forall j \in \{1, \ldots, M\} \tag{4.14}$$

Ceci mène à :

$$\frac{C_j}{D_j} = \frac{C}{D} \tag{4.15}$$

$$D_j = D\frac{C_j}{C} \tag{4.16}$$

$$D_j = \begin{cases} D\dfrac{\beta_j A_j}{A} & \text{si} \quad D > 0 \\ D\dfrac{\beta_j F_j}{F} & \text{si} \quad D \leq 0 \end{cases} \tag{4.17}$$

Il est facile de voir que cette formule vérifie également la conservation de la demande. Bien sûr elle n'est appropriée que lorsque A et F sont non nuls, des cas particuliers doivent être implémentés si ce n'est pas le cas. On répartit alors simplement la demande proportionnellement aux β_j.

Répartisseurs génériques

Considérons maintenant un Répartisseur avec N Fournisseurs $\{(D_i^p, F_i^p, A_i^p)\}_{i\in\{1,\dots,N\}}$ et M Accesseurs $\{(D_j^a, F_j^a, A_j^a)\}_{j\in\{1,\dots,M\}}$.

Un Répartiseur générique peut être construit à partir d'un ensemble de Groupeurs et de Partageurs liés entre eux. Il y a N Groupeurs qui correspondent aux Fournisseurs, et M Partageurs qui correspondent aux Accesseurs. Chacun des N Groupeurs est relié aux M Partageurs par un triplet d'informations $(A_{ij}, F_{ij}, D_{ij})_{(i,j)\in\{1,\dots,N\}\times\{1,\dots,M\}}$. Pour commencer nous allons utiliser deux listes de $M \times N$ poids, (α_{ij}) pour les Partageurs et (β_{ij}) pour les Groupeurs, mais nous verrons plus loin que ces deux listes pourront être combinées en une seule sans changer les résultats.

Pour résumer, les Répartisseurs génériques ont $2MN + N + 2M$ entrées : α_{ij}, β_{ij}, D_i^p, F_j^a, A_j^a, et $2N + M$ sorties à calculer : F_i^p, A_i^p, D_j^a.

Il faut à présent juste reprendre les relations des deux types de Répartisseurs spécialisés précédement décrits. On commence par les Partageurs :

$$F_{ij} = \alpha_{ij} F_j^a \tag{4.18}$$

$$A_{ij} = \alpha_{ij} A_j^a \tag{4.19}$$

Ce qui donne en passant par les Groupeurs :

$$F_i^p = \sum_j \beta_{ij} F_{ij} = \sum_j \beta_{ij}\alpha_{ij} F_j^a \tag{4.20}$$

$$A_i^p = \sum_j \beta_{ij} A_{ij} = \sum_j \beta_{ij}\alpha_{ij} A_j^a \tag{4.21}$$

Puis pour les demandes, en commençant cette fois par les Groupeurs (l'information

circule en sens inverse des disponibilités) :

$$D_{ij} = \begin{cases} D_i^p \dfrac{\beta_{ij} A_{ij}}{A_i^p} & \text{si} \quad D_i^p > 0 \\ D_i^p \dfrac{\beta_{ij} F_{ij}}{F_i^p} & \text{si} \quad D_i^p \leq 0 \end{cases} \tag{4.22}$$

Ce qui en passant à travers les Partageurs devient :

$$D_j^a = \sum_i D_{ij} \tag{4.23}$$

$$= \sum_{i, D_i^p > 0} D_i^p \frac{\beta_{ij} A_{ij}}{A_i^p} + \sum_{i, D_i^p \leq 0} D_i^p \frac{\beta_{ij} F_{ij}}{F_i^p} \tag{4.24}$$

$$= \sum_{i, D_i^p > 0} D_i^p \frac{\beta_{ij} \alpha_{ij} A_j^a}{A_i^p} + \sum_{i, D_i^p \leq 0} D_i^p \frac{\beta_{ij} \alpha_{ij} F_j^a}{F_i^p} \tag{4.25}$$

On voit clairement dans les équations (4.20), (4.21), (4.25) que seuls les produits $\alpha_{ij}\beta_{ij}$ importent en pratique, donc on peut simplifier les expressions en posant :

$$\gamma_{ij} = \alpha_{ij}\beta_{ij} \tag{4.26}$$

ce qui diminue le nombre des entrées.

Au final les relations génériques pour un Répartisseur sont donc :

$$F_i^p = \sum_j \gamma_{ij} F_j^a \tag{4.27}$$

$$A_i^p = \sum_j \gamma_{ij} A_j^a \tag{4.28}$$

$$D_j^a = \sum_{i, D_i^p > 0} D_i^p \frac{\gamma_{ij} A_j^a}{A_i^p} + \sum_{i, D_i^p \leq 0} D_i^p \frac{\gamma_{ij} F_j^a}{F_i^p} \tag{4.29}$$

Bien sûr, dans le cas $N = 1$ on retrouve le Groupeur, et dans le cas $M = 1$ le Partageur... Un Répartisseur générique est représenté figure 4.4.

Un corollaire intéressant de ce mode de construction du Répartisseur générique est que n'importe quel agencement de Groupeurs, Partageurs et Répartisseurs sans boucle (autrement dit on ne relie pas un Accesseur d'un Répartisseur à l'un de ses Fournisseurs) peut *in fine* se représenter par un seul Répartisseur générique. Ce point peut avoir son importance à terme dans les cas les plus complexes. Il est souvent intuitif de travailler avec les Partageurs et les Groupeurs qui sont plus simples à paramétrer, ou plus facile de décrire les interactions avec les ressources grâce à plusieurs couches de Répartisseurs. Mais du point de vue de la charge de calcul, ce mode de représentation peut être très inefficace car compliquant la tâche du Manager. Il pourrait être intéressant de développer une sorte de traduction en un seul Répartisseur générique ayant le même comportement que l'assemblage, par un calcul des coefficients γ_{ij}.

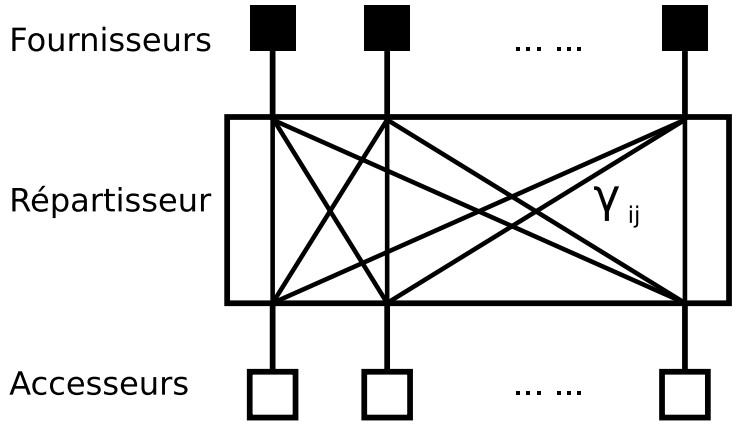

Fig. 4.4 : Illustration d'un Répartisseur générique. On n'a pas individuellement représenté les coefficients γ_{ij} pour ne pas alourdir trop la figure.

La modélisation des ressources n'est d'ailleurs sans doute pas le seul domaine où l'on peut faire cette observation. La hiérarchisation du système peut être une façon efficace de le décrire, mais son implémentation directe peut donner une simulation moins efficiente.

4.1.4 Généricité de l'approche

Il est utile de noter que toutes les propriétés des Répartisseurs découlent *in fine* des caractéristiques spécifiées au niveau des Conteneurs. Les motifs d'entrées-sorties sont imposés par le comportement des Conteneurs, et les relations dans les Répartisseurs viennent en partie des calculs de temps de fonctionnement qu'on peut effectuer au niveau des Conteneurs.

Les hypothèses fortes de modélisation (au niveau bio-physique) sont sur les Conteneurs. En particulier les limitations ne sont que sur le niveau de la ressource (minimum, maximum), mais pas sur l'amplitude des flux. C'est le comportement le plus simple que l'on puisse imaginer, mais cela pose des limites sur les modes de compétition simulables. Des Modèles ne peuvent plus entrer en compétition que par l'épuisement accélérée des ressources, donc un Modèle qui ne tient pas compte du niveau de la ressource ne verra pas de compétition.

Pour avoir une modélisation plus fine de la compétition, il faudrait probablement introduire une notion de limitation sur les flux des ressources. Cela alourdirait bien sûr la simulation mais pourrait à terme s'avérer nécessaire.

Ceci étant, l'approche elle-même est à mon avis générique : définir des Conteneurs qui sont les vrais stocks des Ressources, puis une hiérarchie de Répartisseurs permettant de regrouper ou partager la ressource, le tout communiquant par un ensemble d'entrées-sorties standardisées et cohérentes. Cela permet, dans le cas du paysage, de gérer les ressources indépendamment des Modèles qui agissent dessus, en particu-

lier en terme de spatialisation. Cette possibilité règle une bonne partie des problèmes rencontrés pendant les tests du prototype 2 de simulateur.

Historiquement, c'est d'ailleurs ainsi que tout a commencé. Les modèles de ressources exposés ici ont été développés les premiers suite à l'observation des prototypes. C'est seulement après que la nécessité d'un formalisme minimal de synchronisation de la simulation, non seulement pour les ressources mais pour toutes les informations circulant entre les Modèles, nous est clairement apparue.

4.2 Modélisation de l'eau dans le sol

Compte-tenu de notre focalisation sur les plantes et leur interaction avec l'environnement biophysique, la modélisation du cycle de l'eau, et plus spécifiquement du comportement de l'eau dans le sol, était indispensable.

Cependant, notre but n'était pas d'étudier spécifiquement un modèle détaillé et pleinement réaliste de sol, mais plutôt d'intégrer certaines notions clés dans un modèle simplifé pour étudier les possibilités d'interaction. En ce sens, cette partie de la thèse décrit seulement un modèle très simpliste et qui serait à développer bien davantage pour des applications pratiques.

4.2.1 Equation de Richards

L'équation fondamentale décrivant l'évolution du contenu en eau du sol dans des conditions non-saturées est l'équation de Richards (introduite dans [Richards(1931)] et reprise par d'innombrables chercheurs depuis, par exemple [Witelski(2005)]). Sa forme mono-dimensionnelle est :

$$\frac{\partial \theta}{\partial t} = \frac{\partial}{\partial z}\left(K(\psi)(\frac{\partial \psi}{\partial z} - 1)\right) \tag{4.30}$$

où θ (sans dimension) est la proportion volumique d'eau dans le sol, ψ(m) est la succion du sol (négative dans des sols non saturés), z(m) est la profondeur (positive vers le bas) et $K(\psi)$(m.s^{-1}) est la conductivité hydraulique du sol.

Le comportement de cette équation dépend des deux fonctions $K(\psi)$ (courbe de conductivité) et $\theta(\psi)$ (courbe de rétention). Plusieurs modèles empiriques de ces courbes ont été proposés pour ajuster le comportement mesuré de sols réels (voir [Kosugi et al.(2002)]). Les meilleurs donnent des équations non-linéaires qui doivent être résolues numériquement avec grand soin.

De notre côté, nous avons choisi le modèle le plus simple, toujours dans le but d'étudier les interactions des modèles plutôt que leurs complexités internes.

On définit d'abord un contenu en eau normalisé θ_e :

$$\theta_e = \frac{\theta - \theta_r}{\theta_s - \theta_r} \tag{4.31}$$

où θ_s est la valeur de θ dans un sol saturé, et θ_r est un paramètre ajusté qui peut être interprété comme le contenu en eau résiduel lorsque le sol est sujet à une dépression extrême.

Inspiré par [Kosugi et al.(2002)], on modélise la courbe de rétention d'eau ainsi :

$$\theta_e(\psi) = \begin{cases} \exp(\lambda(\psi - \psi_d)) & , \quad \psi < \psi_d \\ 1 & , \quad \psi \geq \psi_d \end{cases} \tag{4.32}$$

où $\lambda(\mathrm{m}^{-1})$ est un paramètre ajusté, et ψ_d est appelée valeur d'entrée d'air (*air-entry value*).

On choisit aussi une conductivité propotionnelle à θ_e :

$$K(\theta) = K_0(\theta_s - \theta_r)\theta_e \tag{4.33}$$

L'équation de Richards (4.30) devient alors :

$$\frac{\partial \theta_e}{\partial t} = \frac{K_0}{\lambda}\frac{\partial^2 \theta_e}{\partial z^2} - K_0\frac{\partial \theta_e}{\partial z} \tag{4.34}$$

qui est une classique équation d'advection-diffusion. Ce modèle très simple est celui qui a été utilisé pour le second prototype de simulateur.

Bien entendu ce modèle unidimensionnel peut se généraliser à plusieurs dimensions pour inclure la diffusion latérale de l'eau.

4.2.2 Description compatible avec le formalisme de compétition

La réflexion sur la modélisation de l'eau dans le sol a eu plusieurs conséquences pratiques sur les prototypes de simulateurs.

D'abord elle apporte une justification au modèle diffusif utilisé dans le premier prototype de paysage fonctionnel. La seule imperfection est que ce prototype négligeait complètement l'advection dans le sol. Le second prototype, lui, intégrait l'advection et la diffusion sur la dimension verticale, mais négligeait la diffusion latérale entre les cellules, se reposant entièrement sur le ruissellement pour obtenir ces transferts latéraux. Ceci est justifiable si on considère que les cellules sont bien plus grandes horizontalement que verticalement, ce qui revient à dire que des couches imperméables de sol se situent relativement près de la surface. Négliger la diffusion latérale nous permettait aussi de gagner un temps de calcul assez significatif, car le premier prototype avait clairement fait apparaître le coût important d'un calcul tridimensionnel complet.

L'autre enseignement, plus important dans le cadre de travail de l'équipe, est que le potentiel hydrique est une variable significative utile pour modéliser l'interaction plantes-sol. Il permet de s'affranchir des caractéristiques du sol et de ne se baser que sur ce que « perçoit » réellement la plante. Ceci est détaillé plus loin dans la section 5.6.1.

Pour formaliser cet aspect on a eu besoin de définir un Modèle qui traduit nos entrées-sorties standards pour les ressources en potentiel hydrique. C'est un Modèle

transitoire pur, qui a pour entrées les disponibilités en ressources A et F, le paramètre λ du sol, et qui a pour sortie le potentiel hydrique ψ. En négligeant la valeur d'entrée d'air, qui est généralement petite, on trouve en inversant l'équation (4.32) :

$$\psi = \frac{\log(\theta_e)}{\lambda} \tag{4.35}$$

Il ne reste plus qu'à calculer θ_e, ce qui est facile à partir des valeurs A et F :

$$\theta_e = \frac{A}{A+F} \tag{4.36}$$

Soit au final :

$$\psi = \frac{\log\left(\frac{A}{A+F}\right)}{\lambda} \tag{4.37}$$

C'est ce Modèle qui a été utilisé pour les illustrations détaillées au 6.1 et 6.3. Il rend possible le couplage entre les Modèles de ressources et de compétition décrits dans ce chapitre et le Modèle de plante détaillé dans le chapitre suivant.

5. GREENLAB CONTINU

Dans la section 3.5, nous avons décrit les contraintes que le formalisme de simulation impose aux modèles que l'on veut coupler. Il se trouve que les formulations discrètes classiques de GreenLab ne peuvent répondre à ces contraintes en restant réalistes.

Ce chapitre décrit une nouvelle version du modèle GreenLab adaptée au formalisme de simulation décrit au chapitre 3, son implémentation, et quelques-unes des conséquences pratiques de cette nouvelle description en terme de mode de résolution et de stabilité paramétrique. Un modèle légèrement différent mais partageant les mêmes bases a également été publié dans l'article [Li et al.(2009)].

Dans toute la suite, on entend par « GreenLab classique »l'une des versions à temps discret décrites au 1.1.1.

5.1 Echelles de temps pour les plantes

Il y a principalement deux échelles de temps importantes pour la modélisation des plantes, appelées respectivement *temps thermique* et *temps calendaire*.

5.1.1 Temps calendaire

Le temps calendaire t est l'échelle de temps physique, dans laquelle s'inscrivent naturellement la plupart des processus environnementaux avec lesquels les plantes interagissent. En particulier, la plupart des données climatiques sont disponibles au jour le jour ; les phénomènes physiques tels que la diffusion de l'eau dans le sol sont aussi modélisés dans cette échelle de temps.

5.1.2 Temps thermique

La croissance des plantes fait intervenir le temps thermique τ, mesuré en degrés-jour [Jones(1992)]. Il s'exprime par :

$$\tau(t) = \int_{t_0}^{t} \max(0, T(u) - T_{base}) \mathrm{d}u \tag{5.1}$$

où t est le temps calendaire, T la température, et T_{base} la température de base, en dessous de laquelle les processus physiologiques de la plante sont considérés comme stoppés.

C'est l'horloge interne de la plante, qui pilote en particulier l'organogenèse. Les dates d'apparition des nouveaux organes, irrégulières en apparence, sont en fait régulièrement espacées sur l'échelle des degrés-jour pour des plantes à croissance rythmique. L'intervalle de temps thermique séparant deux évènements consécutifs d'organogenèse est appelée phyllochrone et noté γ.

Cette régularité a amené les modélisateurs à considérer les cycles d'organogenèse comme la discrétisation temporelle naturelle pour les modèles de développement de plantes. Ainsi le modèle GreenLab actuel, par exemple, est issu de simulateurs fondamentalement axés sur le développement de la plante, auxquels ont été ajoutés des éléments de fonctionnement de plus en plus nombreux et précis, mais toujours en conservant le formalisme discret basé sur les cycles de croissance, donc le temps thermique.

Comme décrit dans le chapitre précédent, le couplage avec des modèles externes dans un formalisme de simulation suffisament souple impose de revoir cette hypothèse.

5.2 Description formelle de la plante

Une plante peut bien entendu être décrite sous la forme d'une structure arborescente, donc une forme particulière de graphe [Godin et Caraglio(1998)]. Une façon parmi d'autres de représenter ce graphe est une liste de nœuds incluant à la fois un contenu et une description de la topologie, c'est-à-dire quels nœuds sont reliés à un nœud donné.

Formellement, cela veut dire que l'on considère une plante P comme une liste de n nœuds $(n_i)_{i\in\{1,\dots,n\}}$. Chacun des nœuds est un couple (m_i, t_i) constitué d'un contenu et d'une description topologique, comme détaillé ci-après.

En plus de cette liste, on ajoute à l'état de la plante P le temps thermique τ, qui comme décrit plus haut est nécessaire pour modéliser l'évolution dans le temps.

5.2.1 Contenu d'un nœud

Les descriptions botaniques des plantes utilisent le métamère (ou phytomère) comme unité topologique [Barthélémy et Caraglio(2007), White(1979)], et j'ai conservé cette échelle de description même si d'autres auraient pu être choisies, en particulier on pourrait décrire la plante comme une arborescence d'organes et non de métamères. La description au niveau métamère est cependant mieux adaptée aux modèles existants, donc plus économe : il n'est pas utile de descendre au niveau des organes si de toutes façons ils sont toujours structurés en métamères dans nos modèles.

Les métamères et leur fonctionnement s'inscrivent dans le temps thermique, de sorte que l'on doit leur associer au minimum deux dates : τ_i^b la date d'apparition du

bourgeon qui va donner naissance au métamère proprement dit, et τ_i^a la date d'éclosion du bourgeon à laquelle le métamère apparaît. Ainsi le métamère se comporte comme un bourgeon quand $\tau_i^b \leq \tau \leq \tau_i^a$, et comme un vrai métamère quand $\tau > \tau_i^a$.

À noter que dans ce dernier paragraphe le terme bourgeon est employé simplement par analogie avec la botanique ; en pratique, dans le cas des plantes à croissance continue, on peut modéliser un simple méristème par un bourgeon de masse nulle. Dans le modèle les bourgeons servent à transmettre l'information d'organogenèse de cycle en cycle, mais ils ne servent pas forcément de réserve de biomasse.

La description d'un métamère est réduite à sa plus simple expression. On ne conserve que les masses des différents organes (par exemple feuille, moelle de l'entrenœuds, pétioles, cernes, fruits...), classés suivant leurs compartiments (c'est-à-dire le type d'organe considéré), stockées dans un vecteur de dimension c le nombre de compartiments : $[M_i^1 \cdots M_i^c]$. D'autres données peuvent être ajoutées au métamère pour obtenir un modèle de plante au comportement plus souple et plus complexe : âge physiologique, nombre d'organes différents, etc.

Pour résumer le métamère décrit dans un noeud s'écrit, avec les deux temps et la liste des masses par compartiment :

$$m_i = \{\tau_i^b, \tau_i^a, [M_i^1 \cdots M_i^c]\}$$

5.2.2 Représentation de la topologie

Notre modèle de plante n'utilise que les relations topologiques porteur→porté, ce qui veut dire que la topologie est très simple à conserver sous la forme d'une liste d'adjacence, chaque nœud stockant les indices des nœuds qui lui sont liés. Par convention, le premier nœud de la liste est le nœud terminal, les autres sont les latéraux.

Formellement, la topologie est donc une séquence d'entier donnant l'index du nœud lié dans la liste des nœuds : $(l_i^j)_{j\in\mathbb{N}}$. Par convention l'index 0 est un lien ne pointant sur aucun nœud.

5.3 Fonctionnement du modèle

Nous considérons la croissance de la plante comme un processus continu de production et d'allocation de biomasse, interrompu à des dates spécifiques par les évènements d'organogenèse.

5.3.1 Équation de production

La production de biomasse est principalement fonction de la surface photosynthétique totale S de la plante. Pour calculer cette surface, une fonction PS est définie sur les métamères, qui donne la surface photosynthétique d'un métamère. Il suffit ensuite

de sommer pour obtenir S :

$$S = \sum_{i=1}^{n} PS(m_i) \tag{5.2}$$

Une autre façon de calculer le même résultat est de faire intervenir une autre fonction donnant la surface photosynthétique au-dessus d'un nœud donné. Cette fonction CPS est récursive et définie ainsi :

$$CPS(n_i) = PS(m_i) + \sum_{j\in\mathbb{N},\ l_i^j>0} CPS(n_{l_i^j}) \tag{5.3}$$

Il faut noter que CPS est une fonction d'un nœud et non pas seulement d'un métamère, puisque la topologie est nécessaire pour calculer le résultat. Avec cette définition on a évidement $S = CPS(n_1)$. Cette fonction peut aussi être utile dans certains autres calculs et en particulier l'allocation aux cernes.

Ces formules permettent aux organes de tous les compartiments de participer à la photosynthèse. Cependant l'expression la plus simple ne fait intervenir que la masse d'une feuille et une fonction de vieillissement pour chaque métamère. On a ainsi, en admettant que la masse de la feuille soit stockée dans le premier compartiment :

$$PS(m_i) = \frac{M_i^1}{SLW} a(m_i) \tag{5.4}$$

où SLW est la masse spécifique des feuilles.

La fonction a permet de simuler le vieillissement de la feuille sur un métamère. Dans sa plus simple expression c'est une fonction créneau :

$$a(m_i) = \begin{cases} 1 & \text{si } \tau_i^a < \tau < \tau_i^a + T_a \\ 0 & \text{sinon} \end{cases} \tag{5.5}$$

où T_a est le temps de fonctionnement des feuilles.

Bien sûr une fonction plus complexe peut être choisie. Il est même préférable d'éviter ces discontinuités dans l'évolution de la surface photosynthétique, pour obtenir un comportement plus régulier du modèle. On suppose également que le phénomène de sénescence des feuilles est plus doux que ce créneau dans la réalité ; ceci étant, il est difficile de choisir cette fonction.

Le taux de production de biomasse est ensuite calculé au niveau de la plante entière par une équation analogue à celle pilotant le modèle GreenLab classique :

$$Q(t) = E(t)\alpha S_p \left(1 - \exp\left(-k\frac{S(t)}{S_p}\right)\right) \tag{5.6}$$

où $E(t)$ est un paramètre représentant l'influence des facteurs environnementaux, k est le coefficient d'extinction lumineuse de la loi de Beer, et α, S_p sont des paramètres de la plante.

Ce taux de production de biomasse va ensuite être alloué à tous les organes de la plante, comme décrit dans la section suivante. Le fait que la production de biomasse soit disponible instantanément pour tous les organes de la plante reflète l'hypothèse de *pool commun* du modèle classique.

5.3.2 Allocation

Pour déterminer le taux de croissance en biomasse de chaque organe, il est nécessaire de définir une fonction puits *sink*, qui donne le potentiel d'attraction de la biomasse dans le compartiment o d'un métamère :

$$s_i^o(t) = sink(o, \tau, m_i) \tag{5.7}$$

Le puits varie continûment dans le temps thermique. Comme dans le modèle GreenLab ordinaire, une loi bêta est choisie, bien que d'autres modèles de puits soient envisageables. Deux expressions différentes sont utilisées suivant que le métamère se trouve encore au stade bourgeon ou a éclos. Toutes deux utilisent la forme :

$$s_i^o(t) = \frac{\sigma_o}{\sigma_M(a_o, b_o)} \left(\frac{A}{T_e^o}\right)^{a_o-1} \left(1 - \frac{A}{T_e^o}\right)^{b_o-1} \tag{5.8}$$

où A est l'âge de l'organe, σ_o est la force de puits du compartiment o, T_e^o est le temps d'expansion des organes du compartiment o, et a_o, b_o sont des paramètres de forme de la loi bêta.

Le paramètre σ_o^M est un coefficient de normalisation qui correspond au maximum de la loi bêta utilisée, correspondant à :

$$\sigma_M(a, b) = \left(\frac{a-1}{a+b-2}\right)^{a-1} \left(\frac{b-1}{a+b-2}\right)^{b-1} \tag{5.9}$$

L'âge de l'organe est calculé suivant le stade auquel se trouve le métamère :

$$A = \begin{cases} \tau - \tau_b^i & \text{si } \tau_i^b < \tau < \tau_i^a \\ \tau - \tau_a^i & \text{sinon} \end{cases} \tag{5.10}$$

Il est possible d'avoir des formules plus complexes encore dans le cas des cernes, par exemple. Ces formules font intervenir la fonction CPS détaillée plus haut, car l'allocation aux cernes est fonction de la surface foliaire au-dessus du métamère (loi de Pressler [Deleuze(1996)]). Cela ne change pas fondamentalement la description du modèle ; c'est simplement une autre façon de calculer s_i^o.

La production de biomasse est ensuite allouée à chaque compartiment proportionnellement aux puits, donnant une expression de la dérivée d'un métamère. D'abord la demande totale D est calculée par $D(t) = \sum_{i=1}^{n} \sum_{o=1}^{c} s_i^o(t)$. Ensuite :

$$\frac{\mathrm{d}m_i}{\mathrm{d}t} = \frac{Q(t)}{D(t)} \left[s_i^1(t) \ \cdots \ s_i^c(t)\right] \tag{5.11}$$

La dérivée du temps thermique est également facile à écrire d'après la définition :

$$\frac{\mathrm{d}\tau}{\mathrm{d}t} = \max\left(0, T(t) - T_{base}\right) \tag{5.12}$$

On atteint ainsi le stade où l'on connaît la dérivée de la plante P, vu que toutes les autres variables sont constantes dans un métamère *entre les évènements d'organogenèse* :

$$\frac{\mathrm{d}P}{\mathrm{d}t} = \left\{ \frac{\mathrm{d}\tau}{\mathrm{d}t}, \left(\frac{\mathrm{d}m_i}{\mathrm{d}t} \right)_{i \in \{1,\ldots,n\}} \right\} \tag{5.13}$$

Ce n'est pas forcément évident au premier abord, mais $\frac{\mathrm{d}P}{\mathrm{d}t}$ est une fonction de $P(t)$, en particulier à cause de l'expression de S. Ainsi on a obtenu une équation différentielle qui doit être intégrée proprement, tout en vérifiant qu'on s'arrête bien à temps pour les évènements d'organogenèse.

5.3.3 Organogenèse discrète

Le modèle d'organogenèse est identique à celui des modèles GreenLab GL1 classiques. On parcourt la liste des nœuds, on détecte ceux qui contiennent des bourgeons censés éclore au temps courant ($\tau_i^b = \tau$), et un automate est utilisé pour ajouter de nouveaux nœuds à la fin de la liste, en mettant à jour la topologie.

Si les bourgeons ont des masses non nulles, il convient de les allouer entre les métamères générés. Dans l'implémentation actuelle, cette masse est répartie en fonction des puits à l'âge 0, exactement comme pour répartir le taux de production plus haut, mais en opérant directement sur les masses et uniquement sur le sous-arbre généré par chaque bourgeon (c'est-à-dire en fait tous les métamères de l'unité de croissance généré par le bourgeon). Sans doute qu'une manière plus générique de traiter les réserves de biomasses dans la plante devrait être étudiée et utilisée pour résoudre ce problème. Le comportement actuel se rapproche de celui du GreenLab discret dans le cas d'arbres à croissance rythmique.

5.4 Implémentation

L'implémentation a été réalisée sous la forme de modules C++ indépendants, avec une partie générique de gestion des données organisées en arbre, et une implémentation du GreenLab continu basé sur ces structures.

5.4.1 Structure de données pour les arbres

Les structures de données représentant des arbres ont été pensées pour réduire au maximum l'encombrement mémoire, qui dans l'optique paysage peut rapidement devenir rédhibitoire. Les implémentations C++ classiques des arbres se font grâce à des structures représentant les nœuds, contenant des pointeurs vers d'autres nœuds. Les nœuds sont créés et la mémoire allouée dynamiquement au fil de la croissance de la plante. Ceci à plusieurs inconvénients :

– La fragmentation mémoire augmente. Un nœud contient généralement peu de données, et le surcoût dû à l'allocation dynamique devient significatif. De plus,

une mémoire fragmentée rend les nouvelles allocations plus longues du point de vue du temps de calcul. On perd donc sur tous les tableaux.
- Les représentations du contenu de l'arbre et de la topologie sont entremêlées. Ceci interdit de créer des variantes d'un même arbre avec des contenus de types différents.
- Le parcours de l'arbre se fait forcément de proche en proche, récursivement. Il n'y a pas de façon simple et économique de parcourir tous les éléments successivement, par exemple.

Ces problèmes m'ont conduit à revoir entièrement la façon de stocker les données pour les arbres. La nouvelle implémentation stocke les nœuds dans un conteneur `vector` de la bibliothèque standard C++. Ce conteneur est conceptuellement un tableau, avec des facilités de redimensionnement. Ceci correspond à la liste $(n_i)_{i\in\{1,\dots,n\}}$. La topologie est stockée non plus sous la forme de pointeurs typés, mais simplement sous la forme d'entiers représentant les indices du tableau. La seule allocation dynamique restante est celle de la liste de ces indices pour chaque nœud (elle pourrait être supprimée si besoin était en fixant une taille constante, donc un nombre constant de liens par nœuds). Le parcours séquentiel des nœuds pour tout calcul ne dépendant pas de la topologie est trivial, sans aucun surcoût en espace mémoire.

Il est également facile de définir une classe pour un arbre « allégé », qui emprunte la topologie d'un autre. Au lieu de stocker la liste des nœuds, il ne stocke que la liste des contenus, et se sert des indices stockés dans l'arbre complet quand la topologie est nécessaire. Ceci permet à peu de frais d'augmenter temporairement les données stockées dans un arbre. Ainsi la structure conservant l'état de la plante peut être allégée au maximum et ne stocker que l'état réel, pas toutes les variables intermédiaires servant au calcul. Une application de ce principe est le stockage de la dérivée d'une plante, qui partage nécessairement sa topologie avec la plante elle-même.

Une dernière optimisation est la factorisation de l'arbre. Le modèle de plantes utilisé a en effet la caractéristique notable de produire parfois plusieurs nœuds en tous points identiques. Ceci est géré au niveau de la structure de données en associant à chaque nœud une multiplicité μ, qui représente le nombre de fois qu'il est répété dans l'arbre. Ainsi, lors des parcours séquentiels de l'arbre, on peut tenir compte de cette multiplicité. Par exemple si on veut obtenir la surface photosynthétique de l'arbre il faut modifier la formule 5.2 ainsi :

$$S = \sum_{i=1}^{n} \mu PS(m_i) \tag{5.14}$$

Les parcours récursifs sont inchangés, on a tout simplement plusieurs nœuds qui pointent vers un même nœud factorisé. On peut éventuellement économiser des parcours en notant les nœuds déjà parcourus lors d'un algorithme. Bien qu'indispensable pour obtenir des temps de calcul et occupation mémoire raisonnables, la factorisation n'est pas une nécessité formelle et pourrait d'ailleurs être supprimée de manière partielle ou totale suivant les besoins de modélisation.

On a ainsi toute une hiérarchie de conteneurs génériques (comme pour la bibliothèque standard C++, des templates ont été utilisés). La classe de base a été nommée

`IndexedTree`, en référence au fait que tous les nœuds peuvent être accédés directement par leur index. Sans rentrer ici dans les détails trop techniques, il est utile de mentionner la classe `ShallowIndexedTree` qui permet de créer un arbre partageant la topologie d'un autre, et la classe `SubIndexedTree` qui permet d'extraire un sous-arbre d'un arbre existant.

5.4.2 Utilisation pour le modèle GreenLab continu

L'implémentation du modèle GreenLab continu que j'ai réalisé consiste en une collection de classes que l'on peut grossièrement répartir en trois catégories :

- Arbres utilisant les structure de données décrites précédemment
- Contenus de ces arbres
- Structures regroupant les paramètres de la simulation

L'état de la plante P est représenté par un objet de la classe `Plant`, héritant d'un `IndexedTree` contenant des objets `Metamer`. La dérivée de la plante $\frac{\mathrm{d}P}{\mathrm{d}t}$ représentée par un objet de la classe `DPlant` est de manière analogue un `ShallowIndexedTree` (la topologie est partagée avec la plante) contenant des objets `DMetamer`.

La fonction `continuousGrowth()` de la classe `Plant` renvoie une `DPlant` contenant la dérivée de la plante. C'est bien sûr dans cette fonction que l'on utilise l'équation de production et tout le mécanisme d'allocation du flux de biomasse. En interne, un autre `ShallowIndexedTree` est créé sur la base de la plante, stockant un certain nombre de variables de calcul utilisées en particulier pour traiter l'allocation aux cernes.

La fonction `organogenesis()` de la classe `Plant` réalise la simulation de l'organogenèse. C'est ici que la factorisation intervient. Chaque bourgeon possède une méthode `generate()` qui crée les nœuds sortant du bourgeon éclot. Pour ce faire il passe une description de chaque nœud à créer à la méthode `build()` d'un objet `MetamerFactory`, qui renvoie l'index d'un nœud répondant à cette description. Cet index correspond soit à un nouveau nœud ajouté à l'arbre, soit à un nœud identique déjà existant que l'on réutilise. Pour effectuer cette tâche, la classe `MetamerFactory` conserve en interne un tableau associatif mettant en correspondace les descriptions des nœuds (une chaîne de caractères) et les index correspondants. En pratique seuls les nœuds correspondant à des bourgeons non éclos et de masse nulle peuvent être factorisés, et le processus est totalement transparent vu des bourgeons.

La fonction `timeLimit()` prend en entrée un objet `DPlant` et renvoie le temps Δt_{max} maximum pour lequel on peut faire $P + \Delta t_{max} \frac{\mathrm{d}P}{\mathrm{d}t}$ sans manquer un évènement d'organogenèse. Ceci permet d'intégrer numériquement l'équation différentielle sans imprécision sur la date des évènements d'organogenèse. D'après le fonctionnement décrit pour les plantes il est facile de montrer qu'on a :

$$\Delta t_{max} = \frac{\tau_{org} - \tau}{\frac{\mathrm{d}\tau}{\mathrm{d}t}} \tag{5.15}$$

où τ_{org} est le temps thermique de la prochaine organogenèse. Ce temps Δt_{max} donne une borne supérieure au pas de temps d'intégration.

Finalement, la fonction `update()` prend en entrée un objet `DPlant` et un temps Δt, et met à jour la plante en faisant $P(t + \Delta t) = P(t) + \Delta t \frac{\mathrm{d}P}{\mathrm{d}t}$.

5.4.3 Visualisation

Pour les besoins de test, des fonctions de visualisations primitives ont été écrites. Elles se contentent d'écrire un fichier d'entrée pour le logiciel de rendu POV-Ray. La géométrie des organes est limitée à sa plus simple expression : cylindres pour les entrenœuds et les feuilles, sphères pour les bourgeons. Un angle unique de phyllotaxie et un angle d'insertion permettent de reconstituer les positions des métamères les uns par rapport aux autres.

Bien sûr de nombreux effets manquent pour en faire une visualisation réaliste. D'abord il faudrait introduire des notions géométriques qui n'interviennent pas dans la simulation proprement dite : plagiotropie/orthotropie, mécanique... Ensuite on pourrait utiliser des outils de synthèse d'image et en particulier travailler sur les textures. L'utilisation de POV-Ray comme moteur de rendu, bien que donnant un éclairage très réaliste et une grande facilité de composition des scènes, pose des problèmes de performance. Le simulateur peut très facilement créer un arbre qui comporte trop de métamères pour être entièrement chargé en mémoire.

Cette visualisation simple a néanmoins permis de montrer d'autres avantages du modèle continu et de la représentation utilisée. D'abord, le modèle se prête très naturellement à la réalisation d'animations, avec moins d'à-coups que le modèle discret. En maîtrisant la simulation dans le temps calendaire, on peut facilement avoir des sorties régulières de l'état de la plante à passer au visualisateur. Il est également apparu que le format de stockage de la plante utilisé est suffisant pour la visualiser. Ceci a donné lieu à de nouvelles discussions sur un format de sortie très simple, qui pourrait être commun à plusieurs simulateurs, destiné à communiquer avec les spécialistes de la visualisation. Ce format compact, simple à lire et à écrire, pourrait à terme remplacer les formats plus complexes incluant des informations supplémentaires sur l'organisation hiérarchique et botanique de la plante.

Un exemple d'arbre visualisé est montré figure 5.1.

5.5 Comportements et résolution

Le nouveau modèle de plante suscite une réflexion concernant la résolution des équations. Contrairement au modèle discret dont l'implémentation est totalement directe depuis les équations, on est en présence d'une équation différentielle qui se prête à différentes méthodes d'intégration. Ces schémas numériques ont des propriétés diverses, et doivent être adaptés pour tenir compte de la spécificité du modèle : des évènements d'organogenèse interrompent régulièrement la simulation. Le choix du schéma numérique a aussi un impact sur l'ajustement paramétrique, puisque généralement cela se fait en appelant un certain nombre de fois une fonction de simulation.

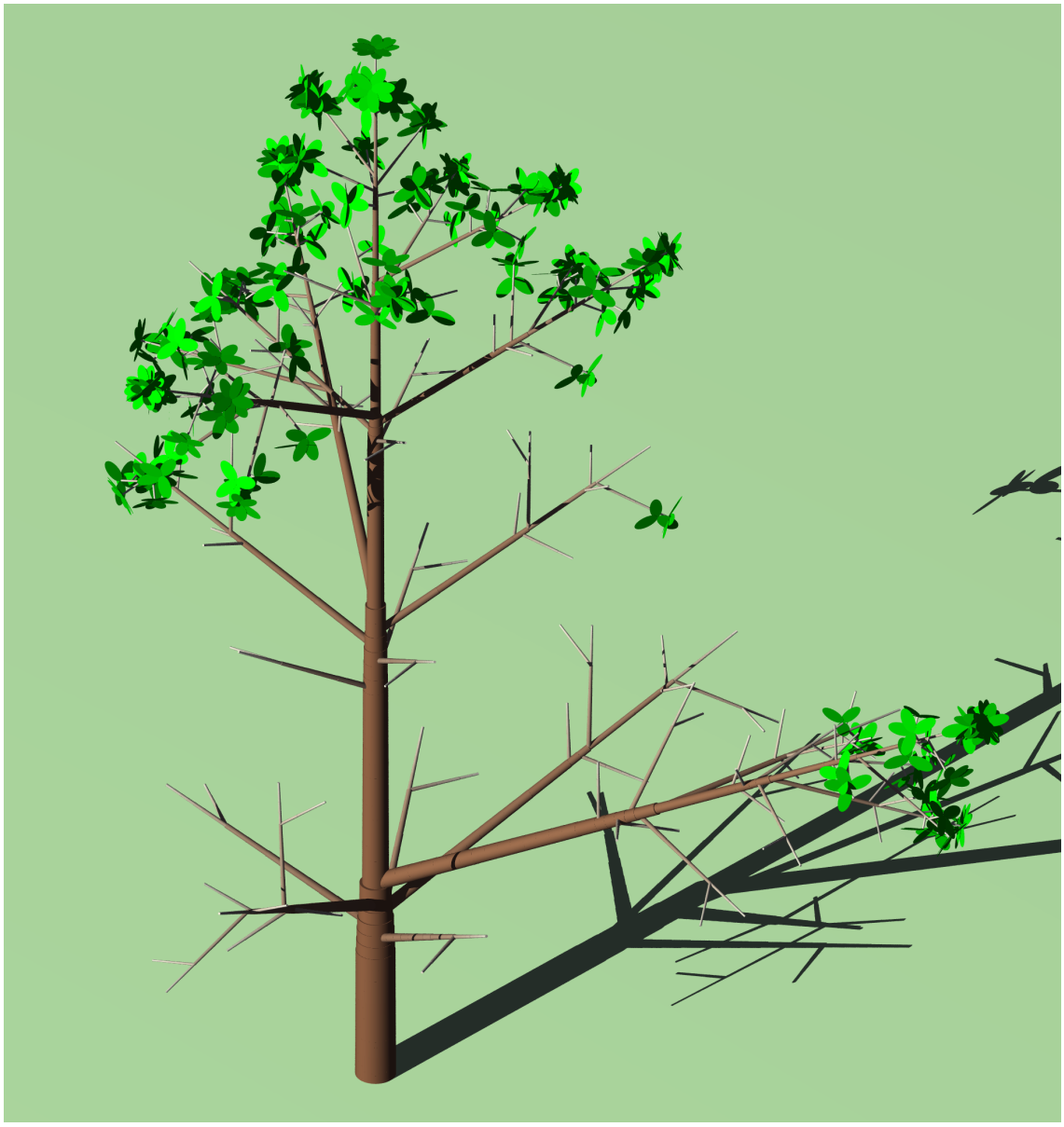

Fig. 5.1 : Exemple d'arbre visualisé avec POV-Ray.

5.5.1 Méthodes d'intégration

Je vais décrire ici deux méthodes courantes pour intégrer numériquement par incrément de temps des équations différentielles. Pour être mieux adapté au problème de la simulation du modèle GreenLab continu, l'équation générique choisie doit représenter deux choses. D'abord une évolution continue, décrite par une équation différentielle (croissance). Ensuite, lorsque certaine bornes sont atteintes, l'état du système doit changer de manière discrète (organogénèse).

Ainsi on part du système :

$$\begin{cases} \dfrac{\mathrm{d}X}{\mathrm{d}t} &= f(X,t) \quad \text{si} \quad s(X,t) \neq b(X,t) \\ X(t^+) &= g(X,t) \quad \text{si} \quad s(X,t) = b(X,t) \end{cases} \tag{5.16}$$

X représentant l'état du système. Les fonctions s et b permettent de donner une borne à l'évolution continue du système. La fonction s fournit une variable synthétique à partir de l'état du système, sur laquelle on va imposer une borne donnée par la fonction b. Lorsque cette borne est atteinte, on modifie l'état par une fonction discrète g.

On impose de surcroît un pas de temps maximal Δt_0 aux méthodes d'intégration.

Méthode d'Euler

La résolution procède par pas de temps Δt suivant l'algorithme classique :

$$x = f(X,t) \tag{5.17}$$
$$\Delta t_l \quad \text{t.q. } s(X + \Delta t_l\ x) = b(X) \tag{5.18}$$
$$\Delta t = \min\left(\Delta t_0, \Delta t_l\right) \tag{5.19}$$
$$X(t + \Delta t) = X(t) + \Delta t\ x \tag{5.20}$$

Notez l'étape 5.18 de calcul du pas de temps où la borne est imposée.

Runge-Kutta d'ordre 2

Plusieurs méthodes ont été implémentées mais seule la méthode dite « midpoint » est exposée ici. L'intégration procède en faisant des allers-retours dans le temps, ce qui impose deux étapes de limitation du pas de temps (5.22, 5.26) pour toujours garder un système cohérent sans dépasser les bornes de la simulation :

$$x_s = f(X,t) \tag{5.21}$$
$$\Delta t_l^s \quad \text{t.q. } s(X + \Delta t_l^s\ x_s) = b(X) \tag{5.22}$$
$$\Delta t_s = \min\left(\Delta t_0, \Delta t_l^s\right) \tag{5.23}$$
$$X_m = X(t) + \tfrac{\Delta t_s}{2}\ x_s \tag{5.24}$$
$$x = f(X_m, t + \tfrac{\Delta t_s}{2}) \tag{5.25}$$
$$\Delta t_l \quad \text{t.q. } s(X + \Delta t_l\ x) = b(X) \tag{5.26}$$
$$\Delta t = \min\left(\Delta t_0, \Delta t_l\right) \tag{5.27}$$
$$X(t + \Delta t) = X(t) + \Delta t\ x \tag{5.28}$$

5.5.2 Différences entre les schémas numériques

Bien entendu le schéma d'ordre 2 est bien plus performant que le schéma d'Euler. Tous deux gagnent en performance quand le pas de temps diminue, mais le schéma « midpoint » est bien plus stable et converge plus rapidement vers la solution explicite. Ceci est vrai pour tout système dont l'état n'évolue pas de manière linéaire, surtout en cas de croissance explosive. En particulier, nos modèles de plantes sont dans ce cas : ils présentent nécessairement une phase de croissance exponentielle, plus ou moins longue, au fur et à mesure que la surface foliaire augmente, jusqu'à saturation de l'équation de production.

Pour illustrer ce fait nous allons montrer les résultats de simulation sur un système à hystérésis très simple. Cet exemple est choisi parce qu'il est plus simple qu'un modèle de plante, peut se résoudre explicitement, et permet d'illustrer les fonctions s et b. L'équation est :

$$X = (u,p) \tag{5.29}$$
$$f(X,t) = (A_p\ u + B_p, 0) \tag{5.30}$$
$$b(X) = H_p \tag{5.31}$$
$$s(X) = u \tag{5.32}$$
$$g(X,t) = (u,\ p + 1 \bmod 2) \tag{5.33}$$

La variable u est l'état continu du système, le signal. La variable p représente le phénomène d'hystérésis et alterne entre les valeurs 0 et 1 chaque fois que la borne est atteinte. Suivant la valeur de p, les paramètres de f A_p et B_p, ainsi que la borne H_p, changent. Ces trois variables peuvent être représentées sous la forme de vecteurs de deux lignes.

Les résultats, obtenus grâce au logiciel Scilab, sont présentés sur la figure 5.2. Comme cette équation est intégrable formellement (la solution est formée par des branches d'exponentielle répétées périodiquement), on peut comparer les solutions trouvées à la solution explicite et constater, comme on pouvait s'y attendre, que la méthode « midpoint » est bien plus performante.

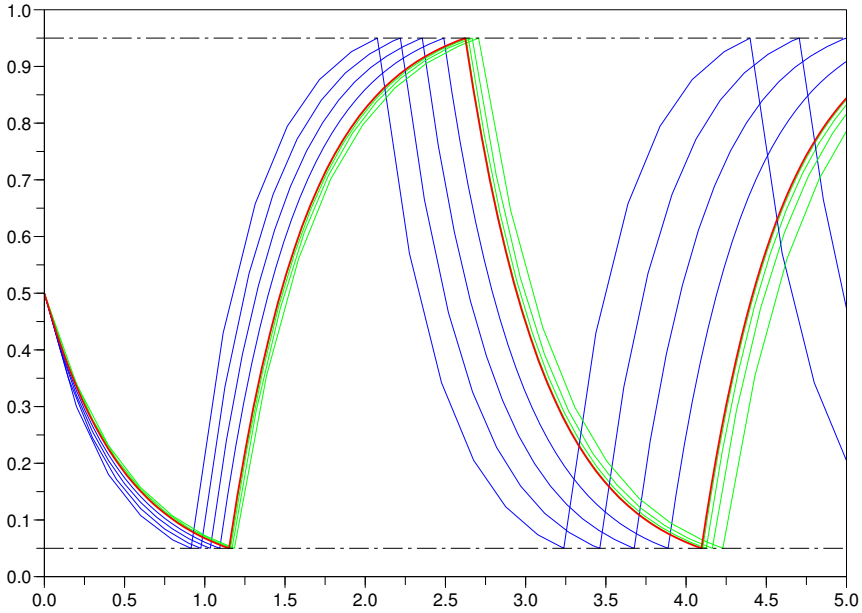

Fig. 5.2 : Simulations du système à hystérésis par les deux méthodes décrites avec des pas de temps maximum différents. On choisit $A_0 = A_1 = -2$, $B_0 = 0$, $B_1 = 2$, $H_0 = 0.05$, $H_1 = 0.95$. Les lignes pointillées représentent les bornes H_0 et H_1. La courbe rouge représente la solution exacte, la série de courbes bleues représente des solutions trouvées par la méthode d'Euler, la série de courbes vertes des solutions trouvées par la méthode « midpoint ». On a pour les deux séries $\Delta t_0 = 0.2, 0.15, 0.1, 0.05$ successivement. Pour les deux méthodes la diminution du pas de temps rend l'intégration plus précise, mais la méthode d'ordre 2 est beaucoup plus performante même pour des grands pas de temps, et se stabilise bien plus vite (on ne distingue même pas la $4^{\text{ème}}$ courbe de la solution exacte). Les dates des évènements sont également affectées par la précision de l'intégration.

Bien sûr la précision a un coût. Il est nécessaire d'appeler deux fois la fonction f, et le couplage entre modèles est bien plus difficile à formaliser quand on veut utiliser une méthode « midpoint ». C'est pour cette raison que l'architecture et le formalisme de simulation précédement exposés sont bien adaptés à une simulation par Euler, et pas vraiment par des méthodes d'ordre supérieur.

5.5.3 Stabilité de l'estimation paramétrique

Un des problèmes clés concernant les modèles de systèmes complexes, tels que les différentes versions de GreenLab, est l'estimation paramétrique. À partir d'observations sur le terrain, on cherche à trouver les valeurs des paramètres permettant de

reproduire ces mesures. C'est bien entendu une étape cruciale, préliminaire indispensable à l'utilisation prédictive des modèles.

Dans les cas les plus répandus, le modèle ne peut reproduire toute la complexité des observations, qui sont de surcroît entachées d'erreurs. On se ramène alors à un problème d'optimisation, qui consiste à trouver le jeu de paramètres permettant de fournir le résultat le plus proche des observations. On fait généralement appel à des méthodes numériques plus ou moins élaborées pour résoudre le problème ainsi posé, depuis les algorithmes de gradient jusqu'aux méthodes heuristiques. Ces méthodes font plusieurs fois appel au modèle de simulation, et donc leur résultat dépend entre autre de l'algorithme de simulation.

Pour les modèles discrets, il n'y a pas de doute sur le choix de l'algorithme, puisque l'algorithme de simulation est exactement équivalent à la description du modèle (du moins si les implémentations sont sans erreurs !). Mais pour des modèles continus, les choix de la discrétisation et de la méthode d'intégration changent les résultats des simulations, et donc peuvent changer les résultats des estimations paramétriques.

Plus formellement, si on note p les paramètres à trouver, la méthode d'ajustement paramétrique renvoie une valeur (probablement différente) $\hat{p}$. Dans le cas d'un modèle continu, cet estimateur n'est pas seulement défini par une méthode d'optimisation, mais aussi par l'algorithme de discrétisation. Il est donc normal que des algorithmes de discrétisation différents fournissent des estimateurs plus ou moins précis.

Une étude complète de la calibration du modèle GreenLab continu reste à faire. Néanmoins, on peut illustrer les effets prévisibles sur une version simplifiée de l'équation de production, avec seulement deux paramètres et sans organogenèse. On produit des mesures artificielles en faisant une simulation avec un très petit pas de temps et par la méthode la plus précise (dans notre cas Runge-Kutta d'ordre 2) que l'on sous-échantillonne, et on cherche à retrouver ces observations en utilisant pour la simulation diverses méthodes d'intégration et pas de temps. Le test a été réalisé grâce au logiciel Scilab ; l'ajustement des paramètres est fait par la méthode `leastsq`.

L'équation choisie est :

$$\frac{\mathrm{d}Q}{\mathrm{d}t} = k_0 \left(1 - \exp(-k_1\ Q)\right) \tag{5.34}$$

où k_0 et k_1 sont les paramètres à estimer. Notez la similitude avec l'équation de production 5.6.

Les résultats sont synthétisés sur la figure 5.3.

La conclusion générale est que la méthode d'intégration la plus précise est aussi celle qui fournit les paramètres les plus précis et les plus stables par rapport au pas de temps, même si toutes les méthodes reproduisent presque parfaitement les observations. Cette stabilité peut être très importante pour une utilisation prédictive du modèle : en effet nos recherches sur le couplage de modèles montrent qu'on ne peut pas toujours faire l'hypothèse d'un pas de temps constant. Ceci souligne aussi l'importance de la poursuite de la réflexion sur l'architecture de simulation : se limiter à des couplages intégrés par la méthode d'Euler nous condamne à des résultats imprécis

et compromet les ajustements des paramètres.

Un autre intérêt d'avoir des paramètres stables par rapport au pas de temps est de faciliter l'interprétation de leurs valeurs. Si certains paramètres ont un sens physique, ou sont mesurables ou estimables indépendamment des observations, un estimateur indépendant du pas de temps sera plus facile à utiliser qu'un autre affligé d'un biais systématique causé par une méthode d'intégration imprécise. Comme on considère des phénomènes biophysiques, il est possible que bon nombre de nos paramètres soient dans ce cas.

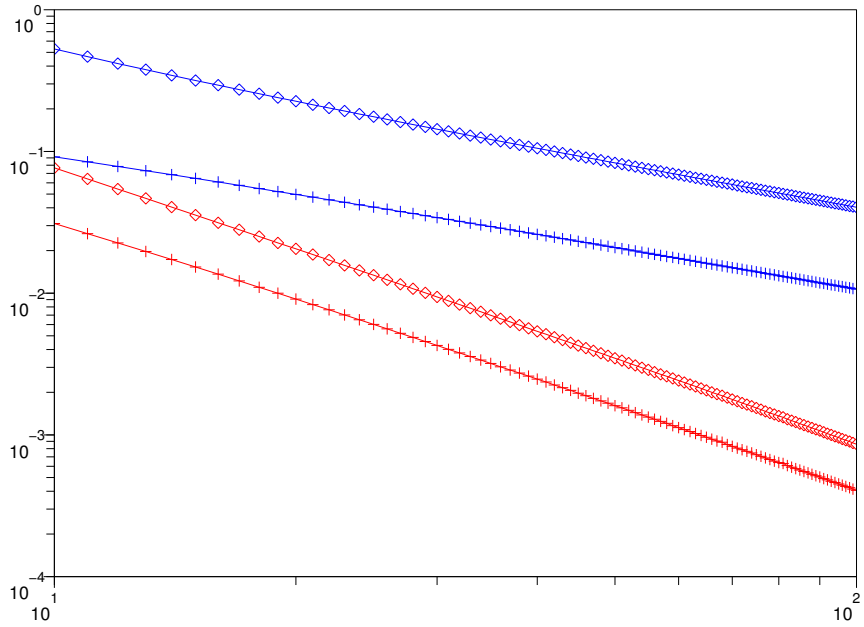

Fig. 5.3 : Ajustements paramétriques pour l'équation 5.34. Les paramètres choisis pour générer les échantillons sont $k_0 = 3$, $k_1 = 2$. L'axe des abcsisses représente le nombre d'échantillons (donc de pas de temps) utilisés pour ajuster les paramètres, l'axe des ordonnées représente la l'erreur commise relativement à la vraie valeur des paramètres. Les courbes bleues sont des ajustements par la méthode d'Euler, les rouges par la méthode « midpoint ». Les ajustements pour le paramètre k_0 sont matérialisés par des croix, ceux pour k_1 par des diamants. On n'a pas représenté la qualité des ajustements (c'est-à-dire la bonne reproduction des observations) car elle est excellente dans tous les cas. En revanche, on voit que la méthode d'ordre 2 est bien plus performante et que l'erreur qu'elle commet diminue plus vite en fonction du nombre de pas de temps.

5.6 Couplage environmental et insertion dans l'architecture de simulation

Cette section décrit comment le modèle GreenLab continu a été inséré dans un Modèle tel que spécifié dans le formalisme de simulation.

5.6.1 Calcul de la variable environnementale E

La première fonctionnalité ajoutée est celle du calcul de la variable environmentale E apparaissant dans (5.6). Nous avons choisi de le traiter comme un calcul réalisé par le Modèle, mais il y avait au moins deux autres solutions : bâtir un Modèle spécifique pour calculer E, ou intégrer le calcul de E dans le GreenLab continu lui-même. La première a été rejetée car les deux Modèles seraient toujours couplés de la même façon et toujours utilisés ensemble. La seconde aurait imposé de manière trop rigide le mode de couplage avec l'environnement, en se restreignant à des entrées fixes. Des cas d'étude ultérieurs pourraient cependant amener à reconsidérer ces choix.

La variable E est modélisée comme le produit de trois facteurs :

$$E = R\ E_T\ E_w \tag{5.35}$$

où R est la radiation photosynthétique incidente, E_T représente l'influence de la température, et E_w l'effet de la réserve d'eau dans le sol.

Le facteur E_w est celui qui permet à la plante de s'adapter au contenu en eau du sol. Le modèle le plus simple pour obtenir cet effet est de le relier linéairement au contenu en eau du sol θ (pourcentage en volume), comme par exemple dans [Wu et al.(2005)]. Cependant, cela ne permet pas de rendre compte des différences de comportement des sols eux-mêmes : l'eau peut être, à un même contenu, plus ou moins difficile à extraire suivant la texture du sol. La variable permettant de s'affranchir de cette dépendance à la nature du sol est le potentiel hydrique ψ, comme décrite dans la section 4.2. Le modèle devient alors :

$$E_w = \frac{\exp(\nu\psi) - \exp(\nu\psi_{min})}{\exp(\nu\psi_{max}) - \exp(\nu\psi_{min})} \tag{5.36}$$

où ψ_{min} est le potentiel hydrique du sol au point de flétrissement, pris ordinairement à $\psi_{min} = -150m$ quelle que soit l'espèce, et ψ_{max} est le potentiel à la capacité au champ, de même $\psi_{max} = -3m$ quelque soit l'espèce. Le paramètre ν est lui spécifique de la plante et représente comment elle s'adapte au contenu du sol. Notez que pour un sol donné, il existe une valeur de ν qui donne la relation linéaire avec le contenu en eau du sol.

Une fonction empirique simple a été choisie pour E_T, présentant un maximum de 1 à une température T_{opt} et atteignant zéro en deux températures extrêmes T_b et T_h :

$$E_T = \left(\left(\frac{T - T_b}{T_{opt} - T_b} \right)^{\frac{T_{opt} - T_b}{T_h - T_b}} \left(\frac{T_h - T}{T_h - T_{opt}} \right)^{\frac{T_h - T_{opt}}{T_h - T_b}} \right)^{\gamma} \tag{5.37}$$

Le paramètre γ contrôle la forme de la courbe et l'acuité du pic. Là encore, un modèle biologique plus élaboré devrait être recherché, mais celui-ci nous suffit pour tester nos capacités de simulation.

5.6.2 Description du Modèle associé

Le Modèle de plantes contient deux variables d'état, accédé à travers des Caches qu'il possède comme décrit plus haut dans le chapitre 3 : l'état de la plante P et sa dérivée $\frac{\mathrm{d}P}{\mathrm{d}t}$. Le temps thermique τ et sa dérivée sont, rappelons-le, inclus dans ces valeurs. Le Modèle a des liens vers des Getters pour les paramètres et les trois variables environmentales : la radiation incidente R, la température T et le potentiel hydrique au niveau des racines ψ. Finalement, le seul Setter (transitoire) est le flux d'eau prélevé dans le sol W.

Nous allons d'abord détailler le contenu de la fonction *trans*.

Elle commence par l'appel de la fonction d'organogenèse qui crée les nouveaux métamères si nécessaire. Puis elle fait le calcul de la valeur de E. Ensuite, elle l'utilise ainsi que l'état P pour calculer $\frac{\mathrm{d}P}{\mathrm{d}t}$. La valeur de W est prise proportionnelle au taux de production de biomasse (défini par l'équation (5.6)) :

$$W(t) = \frac{Q(t)}{wue} \tag{5.38}$$

où *wue* est l'efficience hydrique de la photosynthèse.

Cette équation, couplée à celle définissant E_w (5.36), décrit un modèle de fonctionnement pour les racines. Grâce aux Modèles pour la compétition et les ressources décrits dans le chapitre 4, on pourrait définir des modèles spatialisés de racines comme dans [Varado et al.(2005), Vrugt et al.(2001)], par exemple.

La fonction *cumul* réalise un pas d'intération par la méthode d'Euler :

$$P = P_{prev} + (t - t_{prev}) \frac{\mathrm{d}P}{\mathrm{d}t} \tag{5.39}$$

Enfin, la fonction *time* calcule la date du prochain évènement d'organogénèse, sous l'hypothèse d'une température constante, en utilisant l'équation (5.15).

La nouvelle version continue de GreenLab, implémentée dans un Modèle, constitue un composant important qu'il fallait développer pour répondre à nos objectifs de simulation de l'interaction avec les ressources. Elle permet de tester de manière plus complète l'architecture de simulation. Le chapitre suivant présente en particulier deux systèmes simulés comprenant un ou des Modèles GreenLab continu.

6. ILLUSTRATIONS ET EXEMPLES

Ce chapitre présente trois illustrations des concepts précédement décrits aux chapitres 3, 4 et 5 à la simulation de systèmes dont l'on pourrait retrouver certaines caractéristiques dans un nouveau « paysage fonctionnel ».

Le but n'est pas ici le réalisme quantitatif des résultats mais plutôt l'exploration des capacités simulatoires. Il ne faut donc pas s'étonner de trouver des systèmes vraiment simplistes, voire irréalistes. Les systèmes présentés visent uniquement à démontrer les capacités de l'architecture de simulation et des Modèles développés.

6.1 Plante et eau

La première application que nous allons décrire met en relation les Modèles de plante et de ressource dans un système non spatialisé, mais pouvant montrer des dynamiques complexes.

6.1.1 Description du système

Le système « plante et eau » comprend en fait trois composants principaux :

- un Conteneur symbolisant l'eau contenue dans le sol
- une plante consommant l'eau du sol
- une pompe d'irrigation amenant de l'eau dans le sol

Les Modèles du conteneur et de la plante ont déjà été décrits aux chapitres précédents. La pompe est nécessaire pour éviter l'épuisement trop rapide de l'eau du sol, et c'est surtout le couplage pompe-plante par l'intermédiaire du sol qui crée la dynamique sur cet exemple.

Le Modèle de pompe doit être décrit plus en détail pour bien appréhender le fonctionnement du système. Son principe est de fournir de l'eau à un taux fixe K_p, et de s'arrêter lorsque le stock atteint un certain pourcentage R_{max} de la pleine capacité. La pompe ne redémarre que lorsque le pourcentage passe au-dessous d'un autre pourcentage R_{min}. Ceci est une modélisation d'une stratégie d'irrigation simpliste fondée uniquement sur le contenu en eau du sol et pas sur l'état de la plante.

Pour implémenter ce comportement le Modèle possède un Accesseur (A, F, D) pointant vers le Conteneur à irriguer d'une part, et en entrée un réel E indiquant une

prévision de l'évolution du contenu (en général, celui-ci doit pointer vers la demande sur le Conteneur) ainsi qu'une série de paramètres indiquant les valeurs de K_p, R_{max}, R_{min}. Le Modèle possède en outre un état booléen a, qui indique si la pompe est en marche ou pas.

Comme le temps n'intervient nulle part dans le comportement (la pompe peut démarrer et s'arrêter à n'importe quel moment), la fonction *cumul* est vide. La fonction *trans* contient une arborescence de décision ainsi organisée :

si a
· **si** $\frac{A}{A+F} \geq R_{max}$
· · $D \leftarrow 0$
· · $a \leftarrow faux$
· **sinon** $D \leftarrow -K_p$
· **fin**
sinon
· **si** $\frac{A}{A+F} \leq R_{min}$
· · $D \leftarrow -K_p$
· · $a \leftarrow vrai$
· **sinon** $D \leftarrow 0$
· **fin**
fin

◇

La fonction *time* calcule la valeur t_n de la façon suivante :

si $E \neq 0$
· **si** a
· · **si** $E < 0$, $t_n \leftarrow t_{prev} + \frac{1}{E}(A - R_{max}(A+F))$
· · **sinon** $t_n \leftarrow \infty$
· · **fin**
· **sinon**
· · **si** $E > 0$, $t_n \leftarrow t_{prev} + \frac{1}{E}(A - R_{min}(A+F))$
· · **sinon** $t_n \leftarrow \infty$
· · **fin**
· **fin**
sinon $t_n \leftarrow \infty$
fin

◇

C'est un exemple très simple d'un Modèle générant un évènement provoquant la mise à jour de tout le système. Comme décrit dans la section sur le Manager, les fonctions *cumul* de tous les Modèles au même niveau hiérarchique vont être appelées avant la date fixée par la pompe. En particulier, les Conteneurs vont mettre à jour leur niveau de ressource, et c'est ce changement d'information qui va se propager jusqu'à la pompe et provoquer l'appel de sa fonction *trans*.

Le système global couple la pompe, la plante, le Conteneur de sol, un Partageur pour que la plante et la pompe puissent agir sur le même Conteneur, et un Modèle de calcul du potentiel hydrique fonctionnant comme décrit à la section 4.2.2. Un schéma du graphe du système est représenté sur la figure 6.1. Ici la température et l'ensoleillement sont maintenus constants pour pouvoir se concentrer sur l'influence du couplage lui-même.

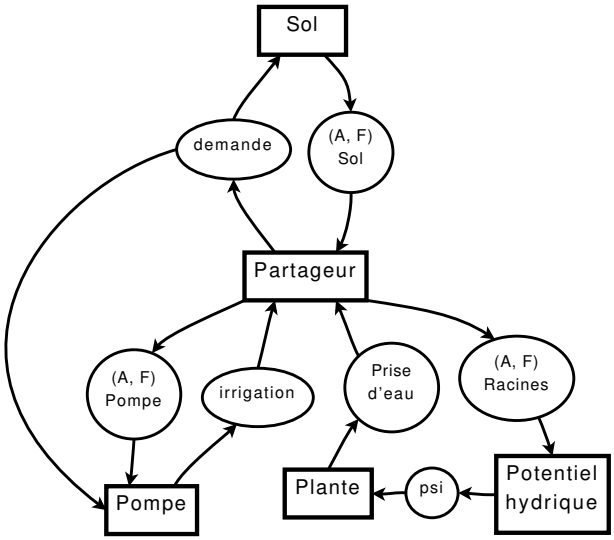

Fig. 6.1 : Système utilisé pour la première application. Comme précédemment, les Modèles sont les rectangles, les Caches sont les ovales. On a regroupé certains des Caches (les disponibilités en ressources) pour gagner en lisibilité, et les Caches contenant des valeurs constantes ne sont pas représentés. On pourrait utiliser la même approche pour coupler davantage de plantes et de Conteneurs, comme on le montrera au 6.3.

6.1.2 Comportement

Pour l'étude du comportement, nous utilisons d'abord un modèle de plante très simple qui émet un métamère unique à chaque cycle de développement, ce qui mène à une simple tige portant des feuilles (cette architecture correspondrait aux modèles botaniques de Corner ou de Holtum suivant la position des organes reproducteurs, non considérés ici). Les valeurs des paramètres sont choisies un peu arbitrairement, nous allons donc nous concentrer sur la description qualitative du comportement.

Des études précédentes du modèle GreenLab isolé dans un environnement constant ont montré qu'un état stable est atteint au bout d'un certain temps, c'est-à-dire que la production instantannée de biomasse se stabilise ainsi que la surface foliaire active [Letort(2008), chap. 4]. Le couplage avec la pompe et le sol fait émerger d'autres comportements asymptotiques. Pour illustrer cela, la figure 6.2 présente plusieurs graphes comprenant 3 courbes : le taux d'irrigation de la pompe (0 quand la pompe est arrêtée), la prise d'eau de la plante (proportionnelle à la production de biomasse), et la surface foliaire active de la plante.

Après des phases transitoires de croissance assez similaires, on peut distinguer deux catégories de comportements asymptotiques :

- lorsque K_p est bas, la prise d'eau par la plante et l'irrigation par la pompe se compensent en régime permanent. La pompe est constamment en marche, la surface foliaire est relativement stable, les petites oscillations visibles ne sont dues qu'à la mort brutale de certaines cohortes de feuilles. Ce cas est représenté en haut de la figure 6.2.
- lorsque K_p est grand, la prise d'eau ne peut compenser la forte irrigation donnée par la pompe. De ce fait, le point de saturation de la pompe est atteint, la pompe s'arrête et le contenu en eau diminue, ce qui cause une chute de la production de biomasse. Une fois le point bas de pompe atteint, l'irrigation reprend, et la production réaugmente. On a ainsi des oscillations périodiques de la production et de l'irrigation. La surface foliaire oscille également mais avec un déphasage, qui vient de ce que la production est intégrée pour donner la surface foliaire. Ce cas est représenté au milieu de la figure 6.2.

Dans le premier cas, la surface foliaire finale augmente lorsque K_p augmente, dans le second cas c'est le contraire, elle augmente lorsque K_p diminue. Il y a une limite marquée entre le premier et le second cas, à laquelle le système change brutalement de comportement. C'est une illutration parfaite des conséquences du couplage entre des systèmes fortement non-linéaires : une bifurcation apparaît autour de laquelle le résultat de la simulation change dramatiquement pour une gamme très réduite de valeurs de K_p. Ce phénomène est représenté sur la figure 6.3. Le graphe du bas sur la figure 6.2 montre l'évolution du système aux alentours de la valeur limite de K_p : la plante semble être dans le régime oscillant, mais à un instant la prise d'eau compense exactement l'irrigation. À ce moment la production se stabilise et la surface foliaire augmente significativement, puis le système entre dans le premier régime, non oscillant.

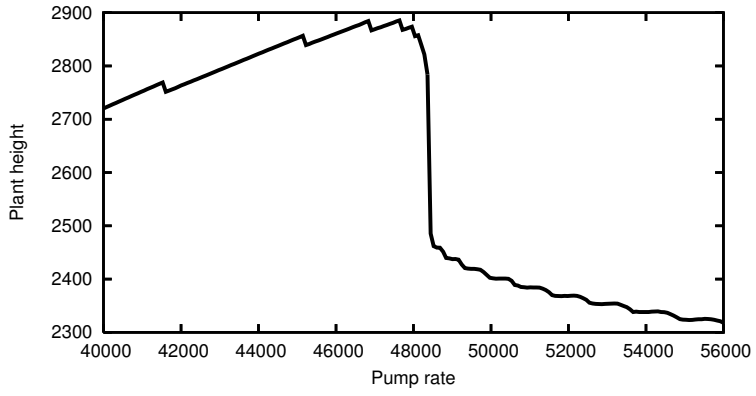

Fig. 6.3 : La hauteur finale de la plante en fonction du taux d'irrigation K_p. Le changement brutal de comportement est clairement visible. Les petites discontinuités pour K_p bas sont dues à la variation du nombre d'impulsion d'irrigation dont la plante a profité aux premiers stades de sa croissance.

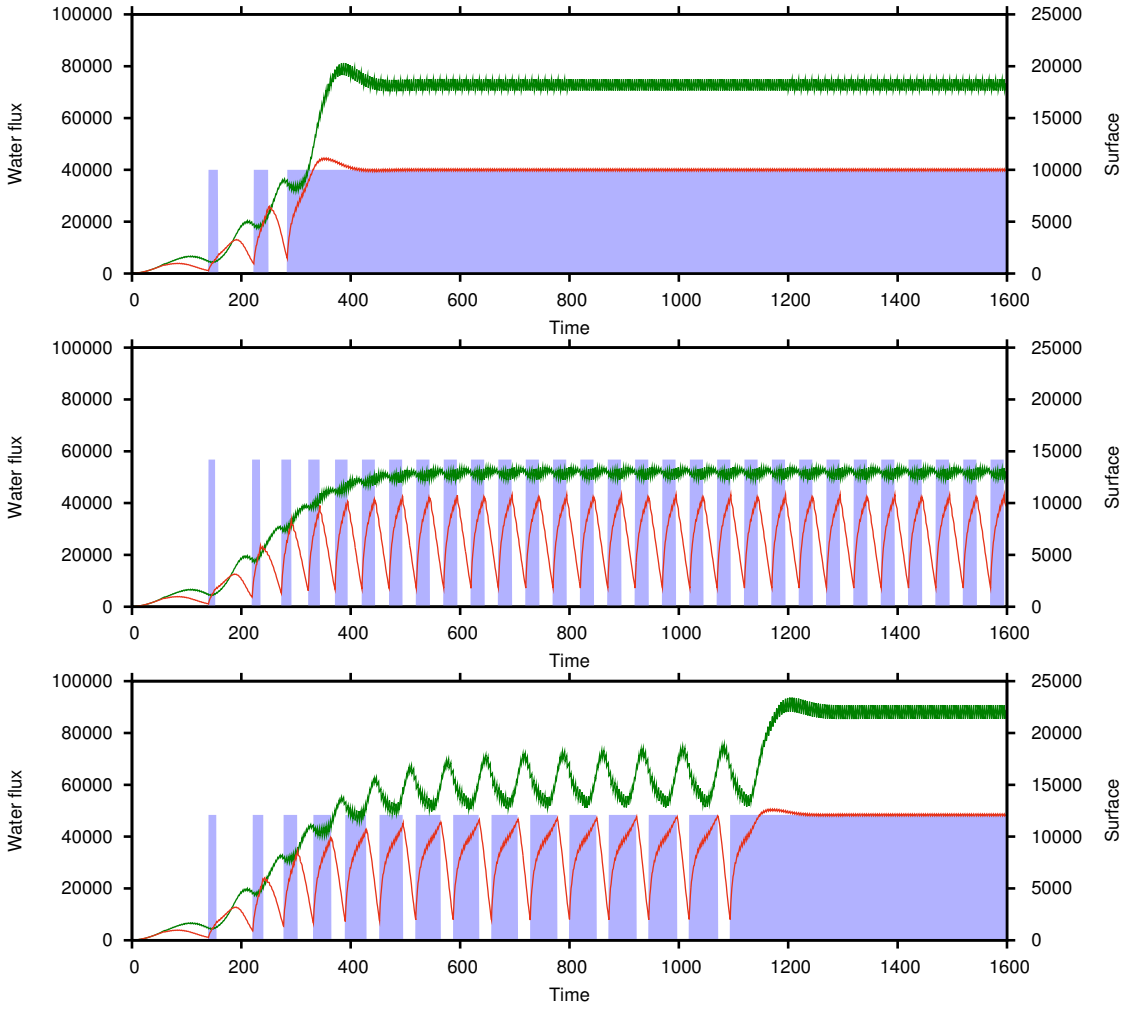

Fig. 6.2 : Plusieurs graphes montrent ici les évolutions possibles du système pour différentes valeurs du taux d'irrigation K_p. La ligne verte est la surface foliaire active, la ligne rouge est la prise d'eau par la plante, les barres bleues représentent les impulsions d'irrigation. Sur le graphe du haut, K_p est bas, en régime permanent la pompe est allumée en permanence et l'irrigation est exactement compensée par la prise d'eau de la plante. Sur le graphe du milieu, K_p est élevé, la plante ne peut prendre de l'eau à une telle intensité et un régime oscillant est atteint. Il y a une bifurcation nette entre les deux types de régime permanent, sur une très faible gamme de K_p. Le dernier graphe correspond à une valeur limite de K_p et montre comment le système passe d'un comportement à l'autre avant d'atteindre un état stable.

La valeur critique de K_p à la bifurcation dépend bien entendu des paramètres de la plante, et aussi du pas de temps primaire de la simulation. Ceci est normal car un pas de temps plus fin permet une meilleure intégration à la fois de la croissance de la plante et de l'évolution de contenu en eau du sol. Les résultats ne changent cependant pas dramatiquement en fonction du pas de temps ; pour un pas de temps de 0.25 la valeur critique est comprise entre 48810 et 49140, pour un pas de temps de 1 elle est comprise entre 48160 et 48490.

Indépendamment du pas de temps imposé par l'utilisateur, on a aussi pu observer sur cette illustration que le système de simulation est capable de détecter les évènements générés par la pompe et insère des pas de temps intermédiaires, à peu près autant qu'il y a d'impulsion d'irrigation. Dans le cas illustré, les paramètres de la plante et la température choisie font que les évènements d'organogenèse coïncident avec des pas de temps imposés, mais il va de soi que si ce n'était pas le cas le simulateur s'adapterait. Par exemple, si le pas de temps imposé est très grand, le simulateur fera de nombreux pas de temps intermédiaires pour permettre à la plante de se développer et à la pompe de s'arrêter ou de démarrer.

6.2 Exemple de spatialisation

Pour des applications futures similaires aux prototypes de paysage fonctionnel (décrits à la section 2.1 et dans le chapitre 2), il était nécessaire d'explorer les possibilités offertes par l'architecture en terme de spatialisation. À cet effet, nous allons maintenant étudier un système très simple faisant appel aux notions décrites au 3.2.3.

6.2.1 Description du système

Le système est cette fois-ci assez abstrait. Il contient une ressource avec une spatialisation raster (conceptuellement, une grille de Conteneurs), et des processus qui la redistribuent. Il y a un processus de diffusion, qui tend à homogénéiser la ressource, et un nombre fixe de processus « pompe » qui transfèrent la ressource d'une région à l'autre.

Les pompes sont ici plus simples que dans la section précédente ; leur comportement est linéaire vis-à-vis des ressources vues en entrée et en sortie. Ce sont des Modèles transitoires purs, sans états, avec juste deux Accesseurs (A_e, F_e, D_e) et (A_s, F_s, D_s) représentant respectivement la ressource en entrée et en sortie de la pompe. La fonction $trans$ calcule D_e et D_s ainsi :

$$p = \min\left(\frac{A_e}{A_e + F_e}, \frac{F_s}{A_s + F_s}\right) \tag{6.1}$$

$$D_e = K_p p \tag{6.2}$$

$$D_s = -K_p p \tag{6.3}$$

K_p étant comme dans la section précédente un paramètre représentant l'intensité de

la pompe. La pompe adapte ainsi son débit aux disponibilités de la ressource en entrée et en sortie.

Le processus de diffusion est un Modèle qui possède un Accesseur spatialisé vers une grille de Conteneurs. La demande sur chaque Accesseur « atomique » est calculée en fonction des flux de diffusion entre les cellules. Un terme correctif classique est appliqué aux flux diagonaux pour minimiser l'impact de la structure raster sur le résultat. Dans la fonction *time* du processus de diffusion, on s'assure que le pas de temps Δt reste inférieur à la limite fixée par la condition CFL, ce qui se traduit par :

$$\Delta t < \frac{\Delta x^2}{D} \tag{6.4}$$

où D est le coefficient de diffusion et Δx est le pas d'espace (taille d'une cellule de la grille). En pratique on assure l'inégalité stricte et la précision en choisissant un pas de temps égal à une fraction fixe de cette limite.

Un Répartisseur particulier est utilisé pour mettre à disposition des pompes et du processus de diffusion les Fournisseurs nécessaires (spatialisés pour la diffusion, non-spatialisés pour les pompes). L'entrée et la sortie de chaque pompe agit sur une aire d'influence carrée, de côté variable en entrée et en sortie, recouvrant plusieurs cellules de la grille.

Un graphe simplifié du système est représenté figure 6.4.

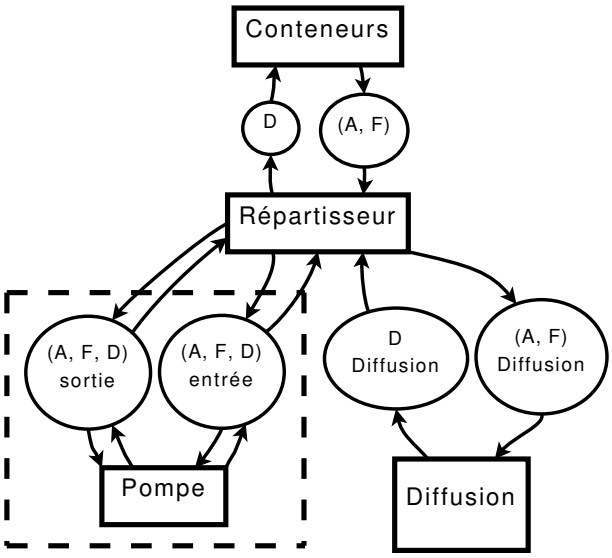

Fig. 6.4 : Système utilisé pour la seconde illustration. La partie encadrée en pointillée est en réalité répétée autant de fois que l'on désire avoir de pompes dans le système. Les Caches compris dans cette partie sont scalaires, non spatialisés. En revanche, les autres Caches sont tous spatialisés, c'est-à-dire qu'ils fournissent une grille de valeurs scalaires.

Pour spatialiser les Conteneurs, nous avons adopté exactement l'approche hiérarchique décrite à la section 3.2.3. Le Modèle Conteneurs spatialisé instancie pour ses calculs un Conteneur « atomique »et pointe ses entrées, ses sorties et son état vers

différents Buffers suivant le point de la grille. Ainsi, le modèle de Conteneur est entièrement réutilisé. Pour le Répartisseur et la diffusion, une approche différente a été suivie. Le modèle de Diffusion ne fait pas appel à une hiérarchie car le modèle « atomique » correspondant n'avait pas été implémenté, et il était également plus facile de travailler directement sur le Répartisseur global que de chercher à réutiliser un Répartisseur existant. En particulier, les coefficients de compétition sont implicites dans cette implémentation particulière.

6.2.2 Visualisation

Une façon un peu particulière de visualiser les résultats a été choisie. À intervalles réguliers, une image est enregistrée, dans laquelle la couleur de chaque pixel représente le niveau de la ressource au point de la grille considéré. Le but était de pouvoir détecter même les très faibles variations de niveau de la ressource pour pouvoir évaluer la précision des algorithmes et en particulier voir les artefacts éventuels de diffusion.

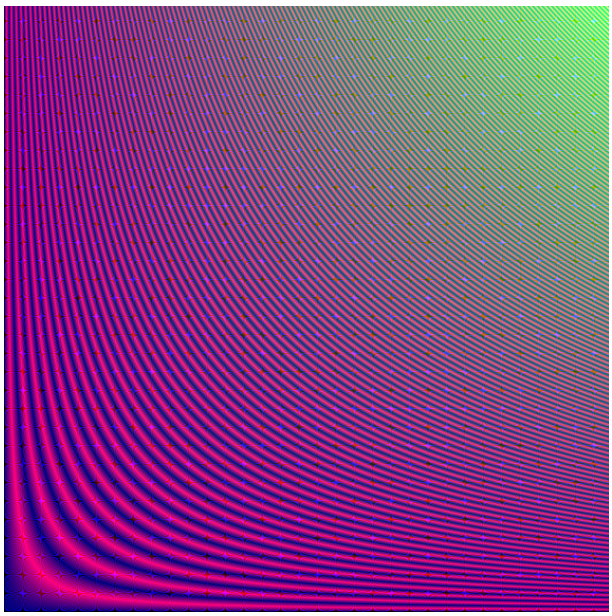

Fig. 6.5 : Exemple du codage utilisé pour visualiser le niveau de ressource. La ressource varie ici suivant le produit xy des coordonnées sur la grille. Notez les franges rouges qui permettent de suivre finement les variations, et les franges bleues qui montrent des artefacts de discrétisation.

Pour tirer parti au maximum de la résolution colorimétrique, le niveau de ressource est encodé sur les canaux rouges, verts et bleus de l'image de la façon suivante. On commence par le transformer en entier non signé, compris entre 0 et $256^3 - 1$, pouvant donc être représenté par 3 octets. Ensuite, on prend l'octet de poids fort, et on l'affecte au canal vert. Puis, suivant la parité de la valeur de cet octet, on affecte au canal rouge soit l'octet de poids médian, soit son complément à 1. Enfin, suivant la parité de la

valeur de l'octet de poids médian, on affecte au canal bleu soit l'octet de poids faible, soit son complément à 1.

De cette façon on obtient une couleur variant continûment avec le niveau de ressource, mais présentant des franges permettant de visualiser des différences très faibles. Un exemple d'image généré par ce procédé est représenté sur la figure 6.5. Les franges forment des sortes de lignes de niveaux qui fournissent une visualisation intuitive du gradient ; cela est utile pour la compréhension des phénomènes diffusifs, qui résultent de flux proportionnels au gradient.

6.2.3 Comportement

Le système présente des comportements émergents moins riches que ceux du système plante-sol, à cause en particulier de la linéarité des Modèles. On vérifie que le système tend vers un régime stable en partant de conditions initiales quelconques, correspondant à de fortes valeurs des ressources aux sorties des pompes, de faibles valeurs à leurs entrées, et des valeurs hors de ces zones reflétant l'équilibre diffusif. La figure 6.6 montre des étapes d'une simulation sur une grille 600x600, avec 5 pompes dont les entrées et sorties ont été réparties au hasard.

Les valeurs des flux de chacune des pompes convergent vers des valeurs différentes, suivant les interactions entre leurs entrées et leurs sorties. Le graphe de la figure 6.7 montre l'évolution dans le temps des flux sortants de chaque pompe.

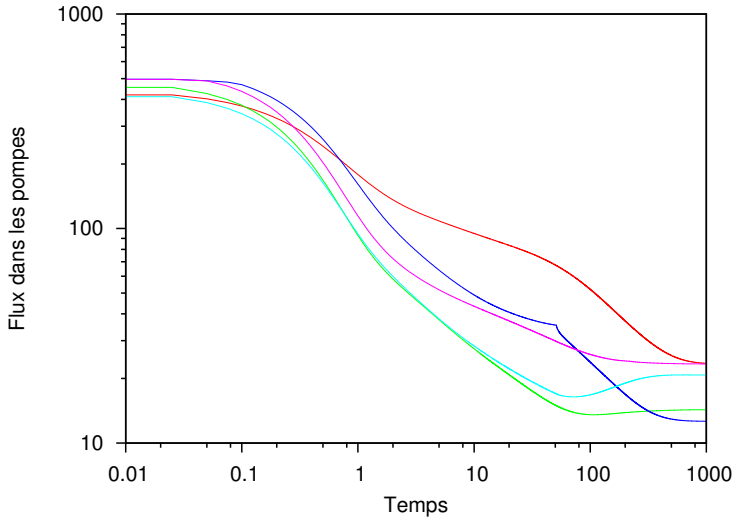

Fig. 6.7 : Flux dans les pompes. On peut voir que chaque pompe se stabilise à une valeur différente. Cela est dû aux interactions complexes non seulement entre les zones d'entrée et de sortie des pompes, mais aussi entre les pompes et le processus de diffusion qui ramène lentement des ressources vers les entrées et les enlève aux sorties. Le décrochement brutal de la valeur d'un des flux est causé par une sorte de transition de phase opérant une jonction rapide entre les zones d'influence de plusieurs pompes.

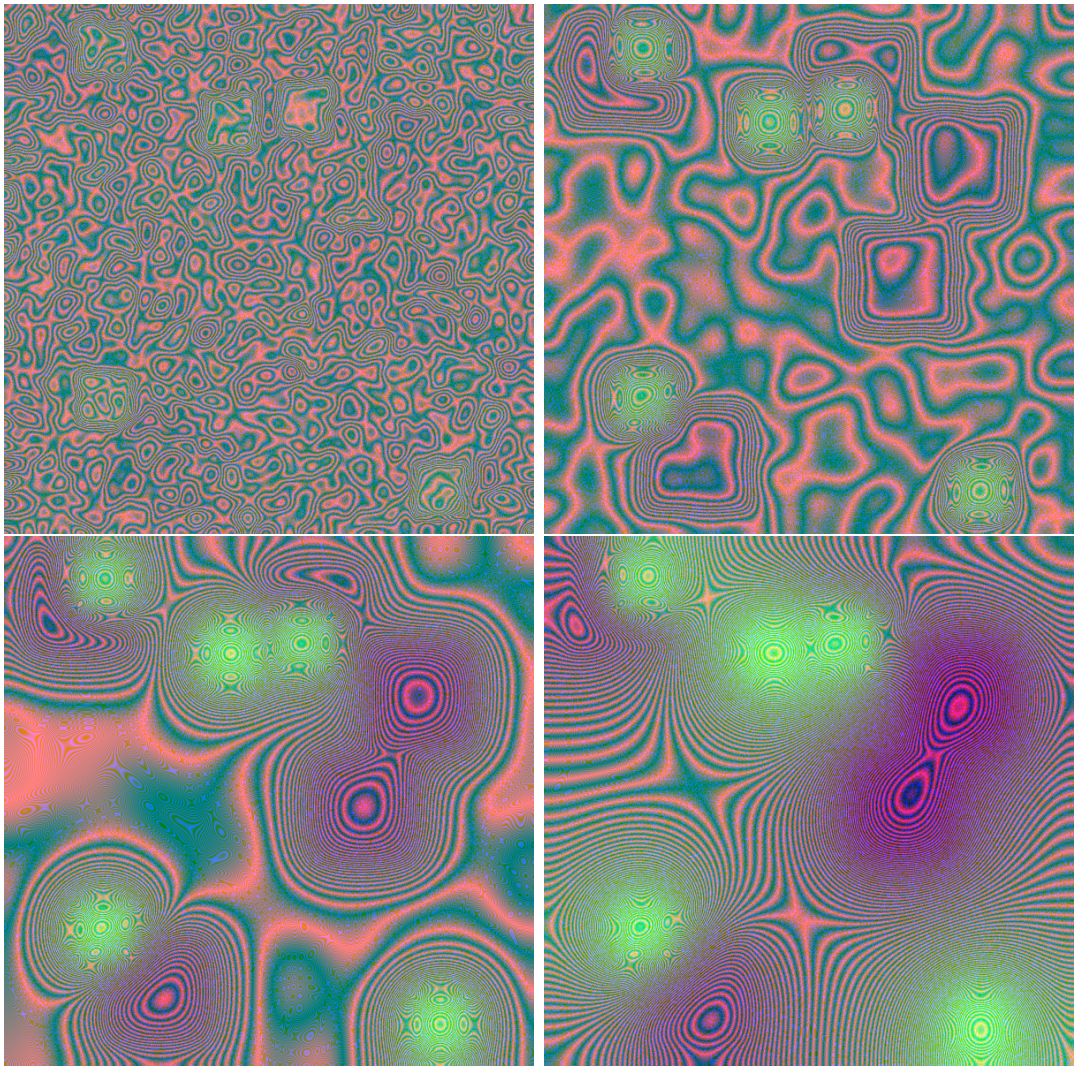

Fig. 6.6 : Évolution du système au fil du temps. L'état est montré ici aux temps 0.05, 0.25, 1.1 et 5. La condition initiale est une valeur aléatoire à chaque endroit de la grille. On distingue aisément l'apparition des zones d'influence des pompes, et le code couleur permet de distinguer les variations fines créées par la diffusion.

6.2.4 Calcul parallèle

Nous avons également exploré quelques possibilités de calcul parallèle sur cette illustration. Cela était particulièrement tentant dans le cas présent car les calculs des Modèles eux-mêmes paraissaient facilement parallélisables, sans toucher à l'architecture de simulation.

La parallélisation du code a été réalisée grâce à la bibliothèque OpenMP, qui était très bien adaptée au travail sur une unique machine à plusieurs cœurs et ne nécessitait que peu de travail au niveau code. Tous les Modèles spatialisés comportaient au moins une boucle interne pour parcourir la grille, à laquelle on a ajouté les directives OpenMP nécessaires (principalement `#pragma omp parallel for` et quelques sections `#pragma omp critical`) pour distribuer les calculs sur plusieurs fils d'exécution. Les modifications les plus lourdes ont porté sur les calculs de diffusion, qui ont été légèrement réorganisés pour éviter tout risque de situation de compétition (*race condition*). La structure imposée par l'architecture de simulation a facilité ce travail, car elle ne favorise pas la modification directe des données, qui peut justement conduire à des phénomènes de ce genre.

Nous avons utilisé une machine DELL Precision 5400T 64bits pourvue de 8 cœurs Intel Xeon à 2.5GHz, tournant sous Ubuntu 9.10, pour tester le programme compilé avec gcc-4.4.1.

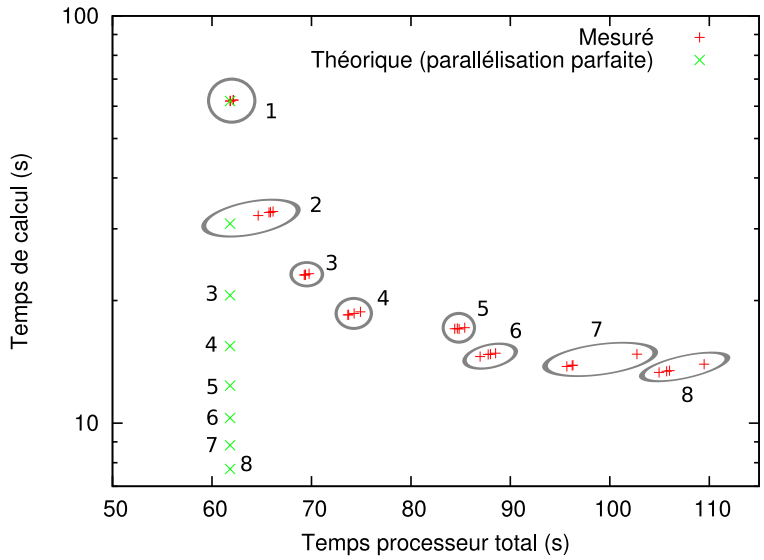

Fig. 6.8 : Évolution des temps de calcul (temps d'exécution du programme) et du temps processeur total consommé (cumulé pour tous les cœurs), suivant le nombre de cœurs utilisés. Chaque groupe de point correspond à un nombre de cœurs utilisé, inscrit à côté de l'ellipse, on a effectué plusieurs tests pour chacun. Idéalement, si la parallélisation était à coût nul, on verrait les points sur une même droite verticale indiquée par les croix vertes. En réalité, on voit que le passage de 1 à 4 cœurs est très bénéfique, mais que l'apport du passage à plus de 4 cœurs est faible.

Comme seules les boucles internes de calcul des Modèles étaient parallélisées, la principale question soulevée était le gain réel apporté. On pouvait redouter que la nécessité de multiples étapes de synchronisation des fils d'exécution ne fasse perdre beaucoup de temps... Le graphique 6.8 montre comment le nombre de cœurs imposé par la variable d'environnement `OMP_NUM_THREADS` affecte les performances.

On voit que l'appel à plus de 4 cœurs est plutôt inefficace, le temps de calcul diminuant peu relativement à la consommation de temps processeur. Ceci est peut-être dû à des problèmes de cache et de mémoire ; la machine ne possède pas un unique processeur 8 cœurs mais 2 processeurs 4 cœurs, le retard pris à partir de 5 cœurs utilisés s'explique sans doute par un problème de communication entre ces deux processeurs. Il est possible qu'une autre méthode de parallélisation, ou une parallélisation de l'architecture de synchronisation elle-même, permette de gagner encore en performance sur des architectures de ce genre.

Cependant le petit travail de parallélisation effectué ici occasionne déjà des gains considérables par rapport à un code non parallèle. Il est donc certain que sur des machines à plusieurs cœurs, cette stratégie doit être employée dans le cas de modèles spatialisés.

6.3 Plusieurs plantes en compétition

L'objet de cette section est d'illustrer les possiblités de l'approche pour simuler des peuplements hétérogènes de plantes en compétition directe sur la même ressource, ce qu'aucun des prototypes n'était capable de faire.

6.3.1 Description du système

Le système est en tout point identique à celui décrit en 6.1, à part qu'on le modifie pour pouvoir gérer un nombre arbitraire de plantes, éventuellement avec des paramètres différents. En pratique cela implique de créer au sein du Manager associé un tableau de sous-modèles, avec les Caches associés. Le Répartisseur a bien sûr plus de Fournisseurs, pour alimenter toutes les plantes... Cette façon de faire est analogue à celle utilisée pour les multiples pompes du 6.2.

Il faut noter que l'effort de développement associé est relativement minime, une fois que l'on a un système à une plante, la principale difficulté étant de rester méthodique et précis au moment de l'établissement des liens entre les Modèles et les Caches... Il est également délicat de trouver un jeu de paramètres donnant des comportements intéressants.

On a choisi de faire au plus simple et de ne pas ajouter de modèle de compétition autre que celui pour la ressource en eau. Il va de soi que l'ouverture sur d'autres modes de compétition (lumière, nutriments en particulier) serait aussi très pertinent pour une étude quantitative plus fine.

Le système ainsi réalisé pourrait remplacer l'une des cellules de terrain du second

prototype de paysage fonctionnel. En effet nous faisions l'hypothèse d'un peuplement homogène de plantes (tant en terme de paramètres de croissance que de date d'apparition) en compétition sur une seule ressource en eau. Grâce aux nouvelles capacités offertes par l'architecture de simulation on pourrait avoir un comportement plus riche à peu de frais.

6.3.2 Comportements

Ce système pourrait se prêter à de multiples études suivant les grandeurs observées et les différences que l'on choisit d'imposer entre les plantes. Nous avons choisi d'étudier la surface foliaire S des plantes et sa variabilité au fil du temps.

Les paramètres des plantes en organogenèse comme en fonctionnement sont choisis identiques, à l'exception d'une variabilité aléatoire affectant le coefficient d'adaptation à l'eau dans le sol ν (voir au 5.6.1, en particulier l'équation (5.36)), et le phyllochrone γ (les évènements d'organogénèse d'une plante sont régulièrement espacés en temps thermique par des intervalles de longueur γ). C'est une première approche pour modéliser la variabilité génétique des plantes. De plus, les temps d'apparition des plantes (temps de levée) sont aussi affectés d'une variation aléatoire. Concrètement, à la création de la plante on retranche une fraction aléatoire de l'âge maximal de la plante à son temps thermique initial τ_0. Ainsi le temps thermique du premier évènement d'organogenèse est retardé d'autant.

Plusieurs scénarios peuvent ensuite être testés. Les simulations suivantes ont été faites sur un groupe de 40 plantes.

- En l'absence de variabilité pour les 3 paramètres ν, γ, τ_0, on retrouve un cas similaire à celui de l'illustration décrite à la section 6.1. Ce cas est rappelé pour mémoire en haut de la figure 6.9. On a choisi des paramètres différents de ceux de la première illustration pour la plante, d'où les différences.
- Une variabilité sur γ résulte en l'apparition d'une légère variabilité de S lorsque les plantes sont en régime permanent. Elle augmente aussi fortement en fin de simulation lorsque certaines plantes commencent à mourir. Dans les deux cas, c'est la mort brutale des feuilles qui explique la variabilité.
- Lorsque ν varie, la variabilité de S augmente progressivement durant la phase de croissance, et plafonne lorsque les plantes sont en régime permanent. Ici c'est la compétition pour l'eau qui est la cause.
- Enfin lorsque τ_0 varie, cela induit une forte variabilité de S en début et fin de croissance. Cet effet masque presque entièrement les deux autres...

Il va de soi que des variations encore plus importantes peuvent être obtenues en augmentant radicalement le décalage de τ_0. On obtient ainsi des comportements assez complexes suivant les hasards de la répartition, comme représenté figure 6.10, où la date de départ peut varier sur une durée double de celle de la durée de vie de la plante. On peut noter l'apparition de zones où la surface foliaire moyenne est élevée, séparée par des zones basses. Il est intéressant de constater que pour une même surface moyenne, l'écart-type peut être plus ou moins fort suivant la phase de la simulation.

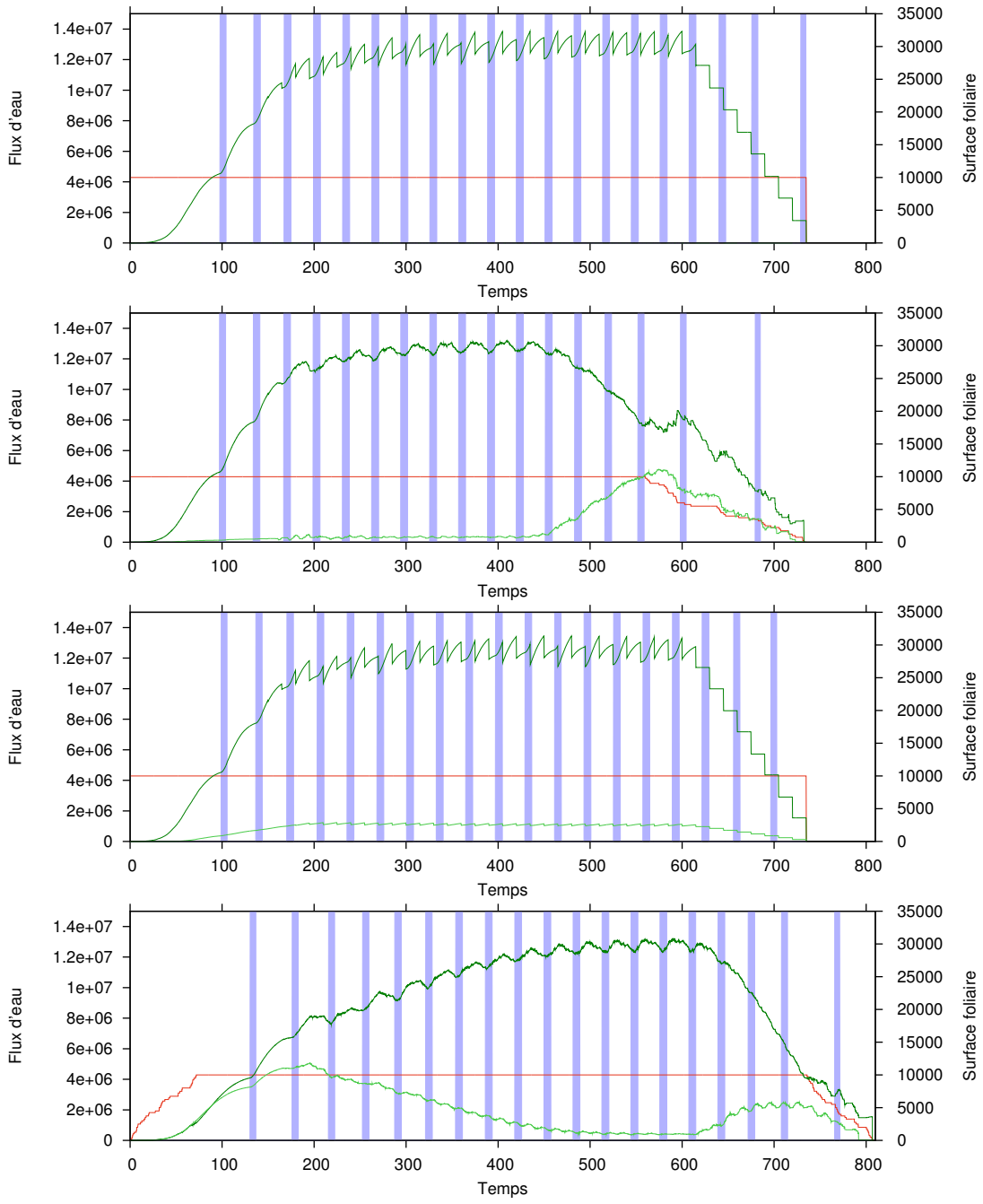

Fig. 6.9 : Impact de différents paramètres sur la variabilité. Les zones bleues matérialisent les périodes d'irrigation, la courbe rouge représente le nombre de plantes actives, la courbe vert foncé représente la surface foliaire moyenne des plantes actives et la courbe vert clair l'écart-type de la surface foliaire. De haut en bas, on trouve le cas où toutes les plantes sont identiques, le cas où le phyllochrone γ varie, le cas où la sensibilité à la réserve d'eau ν varie, et le cas où le temps thermique initial τ_0 varie.

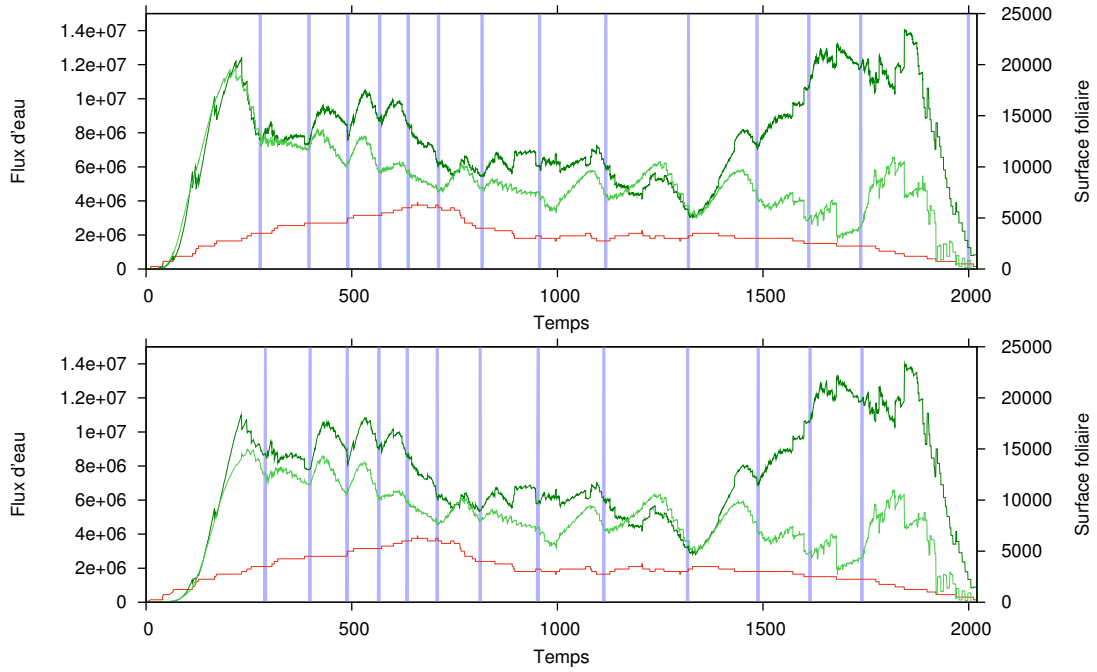

Fig. 6.10 : Simulation sur un peuplement avec de fortes variations de la date de levée. La variabilité de la surface foliaire devient très forte, mais suivant les phases sa corrélation apparente avec la surface moyenne peut changer. Les deux graphes montrent le même système évoluant avec des contraintes différentes sur le pas de temps. Le graphe du haut a un pas de temps contraint à 0.25, alors que le graphe du bas montre l'évolution du système sans contraintes. Sur cet exemple, retirer la contrainte s'avère très bénéfique en terme de temps de calcul (on aboutit à une division par plus de 2), avec une évolution relativement précise. C'est surtout la phase initiale explosive de croissance qui est mal simulée sans contraintes, peut-être à cause de l'intégration par la méthode d'Euler.

Sur cette même simulation, on a fait des expériences sur la gestion du pas de temps. Le graphe du haut est réalisé avec un pas minimum imposé de 0.25 unité de temps, et la simulation prend environ 36 secondes. Le graphe du bas est réalisé sans aucune imposition du pas de temps, c'est-à-dire que ce sont les Modèles eux-mêmes qui le fixent automatiquement. Malgré quelques légères différences, on peut constater la similitude satisfaisante entre les deux cas. Surtout, la seconde simulation est effectuée en 15 secondes seulement... Cela est rassurant sur la robustesse de l'architecture, et rend bien compte de l'intérêt de cette approche.

7. CONCLUSION ET PERSPECTIVES

7.1 Principaux apports

Une des questions soulevées par la problématique de la thèse portait sur les possibilités d'utiliser le modèle GreenLab tel quel dans des simulations en environnement hétérogène à l'échelle du paysage. À l'issue de la thèse, la réponse à cette question a été apportée : il est souhaitable d'étudier une version continue du modèle GreenLab si l'on désire le coupler à d'autres modèles, à quelque échelle que ce soit. Les premiers développements du modèle GreenLab continu sont sûrement un des apports les plus prometteurs de la thèse, et justifient pleinement l'étude au niveau du paysage qui a été faite. Les conséquences nouvelles découlant de cette expression du modèle n'ont pas encore été pleinement explorées et pourraient apporter des bénéfices indépendamment des possibilités nouvelles de couplage qu'elle offre. L'article [Li et al.(2009)] contient une première analyse de ce potentiel. Dans un domaine plus technique, la nouvelle implémentation de GreenLab a aussi permis de dégager des perspectives en terme de visualisation et de communication entre les divers simulateurs de plantes [Jaeger et al.(2010)].

L'architecture de simulation et le formalisme associé sont aussi une avancée significative, qui a fait l'objet d'une publication [Le Chevalier et al.(2009)]. Ils peuvent être utilisés pour de nouvelles applications, et pas seulement au niveau du paysage. Leur devenir futur tient en grande partie aux forces de développement qui y seront consacrées, car, comme décrit dans la section suivante, des raisons techniques font que l'utilisation pratique en est encore un peu fastidieuse. Il faudrait aussi faire des efforts en quelque sorte pédagogiques, apprendre à détecter les cas pratiques où l'utilisation d'un tel formalisme permet de gagner en temps ou en fiabilité, et la proposer aux développeurs concernés.

Comme nous l'avons dit, dans le domaine de la compétition pour les ressources, la démarche proposée semble générique et s'adapterait sûrement à des descriptions plus complexes de la compétition. C'est cette démarche qui constitue l'apport réel dans ce domaine, plus que les détails de formulation qui devront être adaptés à de nouveaux cas. Cependant la description donnée paraît être le minimum capable de gérer des ressources pouvant saturer ou s'épuiser, et donc constitue une base fondamentale pour des développements futurs.

Enfin, les prototypes de « paysages fonctionnels » ont été valorisés tant à travers des articles [Le Chevalier et al.(2007a), Le Chevalier et al.(2007b)], séminaires et démonstrations publiques, qu'à travers les progrès en modélisation mentionnés ci-dessus. Le composant de visualisation du second prototype est suffisament riche pour conti-

nuer en un développement indépendant. Ces prototypes ont permis de détecter une véritable demande pour ce genre de technologie, et indiquent des voies futures de développement.

Cette thèse a aussi apporté une bien meilleure compréhension des problématiques, et en particulier des connaissances plus profondes en ce qui concerne les travaux existants en matière de simulation. Il est certain que beaucoup des choix que nous avons faits, surtout au commencement, se sont révélés peu pertinents faute d'avoir eu à l'époque le recul nécessaire. Nous avons en particulier éprouvé les limites de l'approche bottom-up que nous avions choisi [Le Chevalier et Jaeger(2010)].

7.2 Développements possibles

Cette section décrit les différents travaux qu'il serait possible d'engager en se basant sur les réalisations de cette thèse.

7.2.1 Vers une nouvelle implémentation du « paysage fonctionnel »

Le temps a manqué pour réaliser une nouvelle implémentation qui aurait succédé aux prototypes, utilisant les nouveaux développements et formalismes. Cela dit, les applications restreintes que nous avons décrites permettent d'envisager cette réalisation avec une certaine sérénité. Nous allons décrire ici le mode d'emploi qui permettrait de mettre en place un nouveau « paysage fonctionnel »...

Structuration des conteneurs et processus

Commençons par mettre en place les Conteneurs, puisqu'il sont la base de notre représentation de la structure du paysage. Je pense les séparer en trois familles :

- le sol : pourvu d'une spatialisation sous la forme d'une grille 3D, représente l'eau contenue dans le sol
- l'eau de surface : spatialisé en 2D, ce Conteneur représenterait la hauteur d'eau « libre » en surface à chaque point du paysage. Il implémente une fonction *trans* spéciale qui simule le ruissellement en ré-équilibrant les niveaux de ressource suivant l'altitude du terrain, et dispose donc d'un Conteneur symbolisant l'eau sortie par ruissellement du terrain
- l'atmosphère : un Conteneur simple (on pourrait parler de spatialisation 0D) qui permet de fermer le cycle de l'eau

L'idée est que la somme des contenus de ces trois familles donne toujours la même valeur, ce qui permet de s'assurer du bon fonctionnement du paysage et de la conservation de la ressource en eau. Autrement dit l'eau est transférée d'un Conteneur à l'autre mais ne peut jamais sortir du système.

Sur ces conteneurs on ferait agir un certain nombre de processus :

- précipitations : ce processus prend de l'eau à un accesseur 0D et la verse à un accesseur 2D, suivant la valeur des précipitations spécifiée en entrée du programme
- infiltration : ce processus prend de l'eau à un accesseur 2D et l'infiltre dans un autre accesseur 2D, en fonction du contenu en eau dans ce dernier
- advection-diffusion : ce processus réalise une simulation de l'équation de Richards sur un accesseur 3D
- croissance : ce processus prend de l'eau dans un accesseur 2D et la transpire dans un accesseur 0D, en fonction des résultats de simulations de croissance dans chaque cellule réalisées par le GreenLab continu

Ces processus sont bien sûr reliés aux conteneurs approprié par des Répartisseurs appropriés. On a ainsi les liens suivants :

- atmosphère → entrée des précipitations
- eau de surface → sortie des précipitations
- eau de surface → entrée de l'infiltration
- sol → sortie de l'infiltration (avec un Répartisseur spécifique qui ne cible qu'une couche superficielle du sol)
- sol → entrée et sortie de l'advection-diffusion
- sol → entrée de la croissance (avec un Répartisseur spécifique qui ne cible qu'une couche superficielle du sol)
- atmosphère → sortie de la croissance

Cette structuration permet de représenter la compétition entre les processus au niveau de la couche superficielle du sol, tout en conservant l'hypothèse d'un ruissellement instantané. C'est cette hypothèse qui force à l'inclure comme un phénomène transitoire directement au niveau du Conteneur, plutôt que comme un processus comme les autres. Plusieurs modes de déclenchement du ruissellement sont envisageables, sur le niveau de l'eau ou sur un intervalle de temps régulier.

Bien sûr le seul moyen d'être certain que cette façon de modéliser fonctionne est de se lancer dans l'implémentation puis dans les tests...

Au delà du cycle de l'eau

Nous n'avons pas encore exploré le potentiel d'approches mêlant plusieurs ressources distinctes et des processus agissant simultanément sur elles. Cependant cela semble nécessaire, ne serait-ce que pour des applications sur le stockage du carbone par exemple. Un autre cas intéressant serait de coupler le cycle de l'eau à un cycle des nutriments, grâce auquel les organes et les plantes mortes pourraient être recyclés et assimilés d'abord par la faune, puis par les plantes vivantes. Ce couplage entre deux ressources limitantes serait dynamiquement très riche, mais nécessite bien sûr de compléter les modèles et en particulier celui des plantes.

7.2.2 Formalisme et architecture de simulation

Formalisme

Une des graves faiblesses du formalisme décrit est son orientation implicite vers une méthode d'Euler pour simuler les équations différentielles. Il n'y a pas de formalisation suffisante de l'état d'un modèle et de sa variation, qui permettrait aisément d'utiliser des méthodes de résolution d'ordre supérieur. Comme pour DEVS, il est possible de se servir d'une autre méthode d'intégration au sein d'un Modèle donné. Par exemple, on pourrait réimplémenter un Modèle pour la croissance des plantes utilisant une méthode « midpoint ». Malheureusement, le couplage des modèles ne permettrait pas de coupler deux Modèles résolus par une méthode midpoint et d'obtenir un système résolu par une méthode midpoint. Ceci est dû aux hypothèses faites par le Manager, en particulier l'absence de possibilités d'aller-retour dans le temps puisque les Modèles n'ont pas d'interface permettant cette fonctionnalité.

Un nouveau formalisme fondé sur une description des équations différentielles mixtes comme celles décrites au 5.5.1 devrait être exploré. Il s'adapterait parfaitement aux Modèles déjà décrits, et offrirait une puissance numérique bien supérieure qui permettrait de considérer plus sereinement les problèmes de validation pour nos simulations. En l'état actuel des choses, la simulation est aisée mais probablement imprécise, comme cela a été décrit dans le chapitre 5, et les paramètres seraient donc difficiles à ajuster précisément. Ce problème de précision de la simulation ajoute de la difficulté à une tâche déjà ardue, et le résoudre ou du moins l'alléger est à mon avis indispensable.

Dans le même ordre d'idée, il serait bon d'inclure des informations permettant d'évaluer la validité d'une simulation dans une telle architecture. En particulier, les Modèles devraient pouvoir spécifier des gammes de validité pour leurs entrées et leurs sorties, tant en terme de valeurs qu'en terme d'échelle. On devrait ainsi pouvoir détecter l'appel d'un modèle validé à grande échelle sur un signal évoluant à petite échelle, par exemple. Ceci permettrait de concentrer les efforts de modélisation sur les cas à problèmes...

Si cette information de validation est mise à disposition des Modèles eux-mêmes, on peut imaginer de réaliser des Modèles adaptatifs, qui réagissent par exemple à l'échelle des données entrantes et qui choisissent l'un ou l'autre mode de calcul. On tendrait ainsi vers des possibilités de modélisation multi-échelle renforcées, entre autres bénéfices.

Ingénierie logicielle

La bibliothèque de classes pour la simulation est utilisable en l'état mais quelques-unes des étapes sont fastidieuses. D'abord, les méthodes des Modèles doivent souvent appeler des méthodes des Getters et des Setters pour pouvoir les manipuler. Cela impose un peu de code assez répétitif à écrire. Il devrait être facile, à partir de la description du Modèle et des liens vers les Getters et Setters, de générer un squelette

pour les fichiers de code qui contiendrait déjà toutes ces lignes « triviales ».

Au niveau du Manager, on a le même genre de problème. Le langage C++ n'est pas le mieux adapté pour décrire le graphe des Modèles et des Caches, et il est nécessaire de rédiger beaucoup de code répétitif pour établir ce graphe, décrire l'état du système et lier les Caches aux données. Ceci pourrait être grandement simplifié par des programmes lisant une description plus synthétique du graphe ainsi que des entrées, sorties et états des Modèles, et produisant directement le fichier de code associé au Manager. À plus long terme, on pourrait éventuellement imaginer des interfaces graphiques pour faciliter encore tous ces processus. Au final, on obtiendrait ainsi une plate-forme de génération de code pour les systèmes complexes.

Le code actuel est très monolithique, il impose que tous les Modèles soient codés en C++ ou au moins aient une interface en C++, et compile tout le système en un seul exécutable. Il serait souhaitable de réfléchir à orienter davantage le code vers une architecture plug-in, et peut-être de reconsidérer le langage de développement. Les calculs effectués par le Manager sont très légers, et ne nécessitent pas vraiment la rapidité rendue possible par le C++. Un langage comme Python pourrait se révéler plus approprié pour la tâche et rendre la communication avec d'autres langages plus aisée.

Enfin, il faudrait explorer plus avant les possibilités de calcul parallèle offertes par l'architecture et le formalisme. On a montré au 6.2.4 un exemple de parallélisme à l'intérieur même des Modèles, mais la parallélisation de l'appel des Modèles reste à étudier. On pourrait même éventuellement envisager de distribuer le calcul des Modèles sur des machines différentes, grâce par exemple à MPI, et de conserver un parallélisme interne à chaque Modèle grâce à OpenMP. Bien entendu, il faut que la lourdeur des calculs justifie l'effort de développement, mais des cas au niveau paysage avec des modèles plus complexes peuvent peut-être le nécessiter.

7.2.3 GreenLab continu

Bien sûr il manque encore de nombreuses capacités à l'implémentation actuelle du GreenLab continu, mais la plupart sont des détails qui permettent un fonctionnement plus souple et plus fidèle à la réalité. En revanche il reste plusieurs axes de développement majeurs touchant au cœur du modèle.

Calibration

La première chose à faire concernant le GreenLab continu est l'étude de sa calibration. Je ne pense pas que la tâche sera particulièrement difficile, en tout cas pas plus que ne l'est celle de la calibration d'un modèle GreenLab 1. Le modèle est tout aussi souple et avec un nombre de paramètres équivalent, donc devrait se révéler au moins aussi performant en terme de calibration. Bien entendu, d'autres études encore en cours sur les modèles discrets devront aussi être entamées sur le modèle continu, en particulier l'analyse de sensibilité.

Sur le plan de la calibration, le modèle GreenLab continu offre des possibilités nouvelles en terme d'utilisation des mesures. Par exemple, le modèle est capable de prendre en compte et permet d'essayer de reproduire des mesures réalisées à des dates différentes pendant le même cycle de développement, ce dont les modèles discrets sont *a priori* incapables. Le modèle permet aussi de tenir compte du temps exact de mesure au sein du cycle, et la simulation peut s'ajuster pour fournir une valeur à cette date précise. Jusqu'ici ce souci était secondaire, car les mesures réalisées sur les plantes étaient destructives et ne permettaient donc qu'une mesure de l'état final. Mais de nouvelles techniques peuvent à présent donner certains détails par mesures non destructives, et il est bon d'avoir un modèle capable de les traiter.

Fonctionnement

Sur le plan du fonctionnement même du modèle, il y aura probablement beaucoup d'évolutions au fur et à mesure de la progression des études et de l'acquisition de nouvelles données. On peut cependant être à peu près certain que des changements majeurs devront toucher au moins les trois points suivants.

Processus d'organogenèse La modélisation du développement est pour l'instant insuffisament souple dans le GreenLab continu. Le choix de la similitude avec GL1 impose que tous les organes soient générés aux frontières entre les cycles. Mais les arbres, par exemple, peuvent avoir un processus plus complexe d'organogenèse, avec des pauses et des reprises. Le modèle GreenLab 5 actuellement en développement est prévu pour intégrer ces phénomènes, et il serait très pertinent de reprendre l'organogénèse telle que modélisée dans cette version discrète et d'en doter le GreenLab continu.

Rétroaction structure-fonction Le modèle GreenLab 3 et sa rétroaction entre le fonctionnement et le développement de la plante a été brièvement décrit dans le premier chapitre. Compte-tenu de la puissance de cette modélisation, il paraît indispensable de l'adapter au cas continu d'une manière ou d'une autre. Ceci impose d'étudier la caractérisation de la compétition trophique dans le cas du modèle continu. Il me semble certain que cet indicateur devrait être une grandeur intégrée et pas instantanée, différenciation qui n'a pas vraiment de sens dans le cas discret. Cela permet de lisser les effets de variations brutales de l'environnement ou de variations fortes de la demande pendant un cycle. J'avais envisagé une modélisation fondée sur la masse des bourgeons, leur donnant un comportement différent au moment de l'éclosion suivant la biomasse qu'ils contiennent, mais je n'ai jamais implémenté ni testé cette variante.

Nouveaux puits L'expression du puits des compartiments d'un métamère donnée au 5.3.2 est commune dans les versions existantes du modèle GreenLab. Cependant, elle soulève plusieurs questions. Tout d'abord, le temps d'expansion est difficile à mesurer et potentiellement variable dans la plante pour un même type d'organe. Ensuite elle

soulève des questions quand au mode de normalisation de la courbe, par le maximum (comme nous l'avons fait ici), par la valeur de l'intégrale... Enfin, le recours à des courbes empiriques devrait être le moins fréquent possible.

Sur la base de ces constatations, un certain nombre de partenaires du projet GreenLab ont commencé à développer un nouveau mode de calcul de la force de puits, basé davantage sur des arguments biologiques. Pour schématiser, il fait dépendre la force de puits de la masse de l'organe. Il pourrait être pertinent d'inclure ce nouveau mode de calcul dans le GreenLab continu car il introduit encore plus de couplage entre l'allocation et la production passée, donc aboutit à une équation différentielle aux comportements encore plus riches, pour laquelle les modes de résolution d'ordre supérieur sont d'autant plus intéressants.

Visualisation

Le modèle continu n'a pas encore été pleinement exploité en ce qui concerne la visualisation. La possibilité d'obtenir des animations continues sans à-coups (ou en tout cas pas davantage que ne le montrerait la vraie plante) a un réel potentiel pour la communication. Comme le fonctionnement est détaillé en-dessous du cycle, il est possible de modéliser les variations saisonnières des arbres, et un peu de travail en visualisation (sur la couleur des feuilles par exemple) permettrait de gagner en réalisme de représentation.

Le module de visualisation actuel avec POV-Ray est très limité et ne peut pas traiter rapidement des arbres complexes. Mais d'autres visualiseurs sont actuellement développés, et il suffirait qu'ils soient capables de gérer le temps calendaire continu et de lire un format commun de représentation des résultats de simulation pour qu'ils puissent tirer pleinement parti du modèle continu.

On voit donc que les perspectives ouvertes par la thèse sont très vastes et couvrent de nombreux domaines. Ceci doit être vu comme un signe de succès du travail exploratoire inhérent au sujet et au contexte. Les développements concrets réalisés établissent une base solide sur laquelle on peut construire et offrent des possiblités nouvelles qui pourront être exploitées dans de nouveaux projets, et pas seulement dans un contexte paysage.

ANNEXE

A. PROGRAMMATION LITTÉRAIRE

La programmation littéraire, en anglais *literate programming*, est une technique introduite [Knuth(1984)] par Donald E. Knuth (entre autres concepteur du logiciel TEX) visant à améliorer la qualité de la documentation des programmes informatiques.

À la base de cette idée, se trouve la constatation qu'un programme informatique n'est pas vu et compris de la même façon par un ordinateur et par un être humain. Ainsi, pour expliquer les actions exécutées, on peut vouloir aller rapidement à l'essentiel, ou souligner d'abord la structure avant d'entrer dans les détails. Un compilateur ou un interpréteur, au contraire, ne cherche qu'à exécuter les instructions dans l'ordre...

Du point de vue du programmeur, la programmation littéraire est en fait un métalangage, associant un langage de documentation (typiquement LATEX ou HTML), et un langage de programmation (C++, Java, Pascal...). Le programme et sa documentation sont écrits dans un fichier de format spécial, appelé *web*, dans l'ordre naturel d'explication des instructions. Un programme auxiliaire se charge ensuite de générer d'une part un ou des fichiers de documentation, contenant le code et ses explications, et d'autre part un ou des fichiers de code source compilables.

Cette méthode présente par rapport à l'approche classique d'écriture de commentaires dans le code plusieurs avantages. D'abord, on peut tirer parti des énormes possibilités des langages de documentation : mise en forme évoluée, insertion de figures, formules, références croisées... Ensuite, la documentation est lue dans l'ordre logique (du moins celui voulu par le rédacteur...) et non dans celui du code source.

Nous allons voir ici un exemple simple de cette technique, qui met en œuvre les principaux concepts.

A.1 Blocs et fragments

La première notion essentielle est celle de *bloc*. Le code source est décrit sous la forme d'une arborescence de blocs logiques. Voici un exemple de définition d'un bloc, nommé, pour l'exemple, « un bloc » [1] :

⟨*un bloc*⟩ ≡ 1

```
    ici le contenu du bloc
```

◇1, 2, 4 ; 5

[1] ne prêtez pas attention pour le moment aux nombres qui apparaissent à droite et après le bloc, ils seront expliqués dans la section *Références croisées*, p. 118

Afin de donner plus de souplesse à la description, les blocs peuvent être décrits en plusieurs *fragments* séparés. Une extension d'un bloc est repérée par le signe + ≡ qui remplace le simple ≡.

⟨*un bloc*⟩+ ≡ 2
    la suite du bloc

◇1, 2, 4 ; *5*

Un bloc peut être référencé dans un fragment. Cela signifie que le texte du bloc devra être inséré à l'endroit de cette référence pour reconstituer le code source :

⟨*un autre bloc*⟩ ≡ 3
    ceci est le texte
    d'un autre bloc

◇3, 7 ; *4*

⟨*un bloc*⟩+ ≡ 4
    insérons un autre bloc
    · ⟨*un autre bloc* 3⟩
    le bloc peut se poursuivre ensuite...

◇1, 2, 4 ; *5*

L'indentation sera respectée lors de la génération du code source, le contenu de « un autre bloc » sera donc indenté par rapport au contenu de « un bloc ».

Certains blocs particuliers définissent un fichier de sortie. Par exemple définissons le fichier *un.txt*. Le nom est automatiquement mis en forme de manière particulière pour bien signaler un fichier.

⟨*"un.txt"*⟩ ≡ 5
    une ligne
    ⟨*un bloc* 1⟩
    la fin du fichier

◇5 ; *6*

Bien sûr, les fichiers se comportent comme des blocs normaux, ce qui permet par exemple de les regrouper, eux aussi, si besoin est :

⟨*Fichiers*⟩ ≡ 6
    ⟨*"un.txt"* 5⟩

◇6 ;

Parfois, le code source contient des passages longs et pas très intéressants (entrées-sorties, définitions de constantes, fonctions similaires...). Pour alléger les explications,

et éviter des longueurs, on peut si besoin est abréger le code source dans un fragment. Par exemple :

⟨*un autre bloc*⟩+ ≡ 7
    Attention, passage long :
    [... *Abrégé* ...]

◇3, 7 ; *4*

Le fichier de sortie sera normalement éloquent...

A.2 Fichier généré

Voici au final le contenu du fichier *un.txt* :

◇

```
une ligne
ici le contenu du bloc
la suite du bloc
insérons un autre bloc
        ceci est le texte
        d'un autre bloc
        Attention, passage long :
                        _
          ___ ___    __| | ___
         / __/ _ \ / _` |/ _ \
        | (_| (_) | (_| |  __/
         \___\___/ \__,_|\___|

         _
        | | ___   _ __    __ _
        | |/ _ \| '_ \ / _` |
        | | (_) | | | | (_| |
        |_|\___/|_| |_|\__, |
                          |___/


le bloc peut se poursuivre ensuite...

la fin du fichier
```

◇

Notez comment les blocs sont entrelacés.

A.3 Références croisées

Pour faciliter la navigation dans la description, un système de référence croisées est automatiquement généré.

À droite de chaque définition de fragment, se trouve un numéro unique qui lui est attribué. En dessous de sa définition, se trouve une liste de références sous la forme [*n° de fragment*]–[*page de définition*]. Les références en caractères droits indiquent les fragments qui définissent le bloc. Celles en italique indiquent ceux qui font référence au bloc.

Quand un bloc est référencé dans le corps d'un fragment, la référence qui suit le nom du bloc est celle de son premier fragment.

Normalement, ce système rend la navigation relativement aisée... Un index, représentant l'arborescence des fragments, peut aussi être généré. La forme des références est la même que précédemment :

B. PSEUDOCODE COMPLET DE LA FONCTION DE SYNCHRONISATION

Voici le pseudocode étendu de la fonction *cumul* d'un manager de simulation tel que décrit au 3.2.2.

$t_c \leftarrow t_0$
$t_p \leftarrow t_0$
faire
· $t_c \leftarrow \min(\{time_j(t_p, I_j)\}_j, t_1)$
· nettoyer A et U
· **pour tous** les Modèles dans M
· · $O_j^C \leftarrow cumul_j(t_c, t_p, I_j)$
· · ajoute O_j^C à U
· · **si** l'état du Modèle j a changé
· · · l'ajouter directement à A
· · **fin**
· **fin**
· **pour tous** les Caches dans U
· · **si** le Cache n'est pas cohérent
· · · mettre à jour le Cache
· · · ajouter les Modèles affectés par le Cache à A
· · **fin**
· **fin**
· **tant que** A n'est pas vide
· · nettoyer U
· · **pour tous** les Modèles dans A
· · · $O_j^T \leftarrow trans_j(I_j)$
· · · ajouter O_j^T à U
· · **fin**
· · nettoyer A
· · **pour tous** les Caches dans U
· · · **si** le Cache n'est pas cohérent
· · · · mettre à jour le Cache
· · · · ajouter les Modèles affectés par le Cache à A
· · · **fin**
· · **fin**
· **fin**
· $t_p \leftarrow t_c$
tant que$(t_c < t_1)$

TABLE DES FIGURES

BIBLIOGRAPHIE

[Alweis et Deussen(2005)] M. Alweis et O. Deussen. *Modeling and visualization of symmetric and asymmetric plant competition*. Dans *Eurographics Workshop on Natural Phenomena, Dublin*, p. 83–88. Eurographics, 2005.

[Barczi et al.(1997)] J. Barczi, P. de Reffye et Y. Caraglio. *Essai sur l'identification et la mise en oeuvre des paramètres nécessaires à la simulation d'une architecture végétale : le logiciel AMAPsim*. Dans *Modélisation et simulation de l'architecture des végétaux*, Sciences Update, p. 205–254. Inra édition, 1997.

[Barthélémy et Caraglio(2007)] D. Barthélémy et Y. Caraglio. *Plant architecture : a dynamic, multilevel and comprehensive approach to plant form, structure and ontogeny*. Annals of Botany, tome 99, n° 3, p. 375–407, 2007.

[Bergez et al.(2009)] J. Bergez, P. Chabrier, F. Garcia, C. Gary, D. Makowski, G. Quesnel, E. Ramat, H. Raynal, N. Rousse et D. Wallach. *Record : a new software platform to model and simulate cropping systems*. Dans *Farming System Design*. 2009.

[Birch(2006)] C. P. Birch. *Diagonal and orthogonal neighbours in grid-based simulations : Buffon's stick after 200 years*. Ecological Modelling, tome 192, n° 3-4, p. 637 – 644, 2006.

[Blaise et al.(1998)] F. Blaise, J.-F. Barczi, M. Jaeger, P. Dinouard et P. de Reffye. *Modeling of metamorphosis and spatial interactions in the architecture and development of plants*. Dans T. Kuni et A. Luciani (réds.), *Cyberworlds*, p. 81–109. Springer Verlag, Tokyo, 1998.

[Blaise et de Reffye(1994)] F. Blaise et P. de Reffye. *Simulation de la croissance des arbres et influence du milieu : le logiciel AMAPpara*. Dans *2nd African Conference on Research, Computer Science (CARI94)*, p. 61–75. J. Tankoano, Ed. INRIA, ORSTOM, 1994.

[Bormann et al.(2007)] H. Bormann, L. Breuer, T. Gräff et J. A. Huisman. *Analysing the effects of soil properties changes associated with land use changes on the simulated water balance : A comparison of three hydrological catchment models for scenario analysis*. Ecological Modelling, tome 209, n° 1, p. 29 – 40, 2007. Recent Developments in Hydrological Modelling towards Sustainable Catchment Management.

[Campy et Macaire(2003)] M. Campy et J.-J. Macaire. *Géologie de la surface, Érosion, transfert et stockage dans les environnements continentaux*. Dunod, 2003.

[Chiba et al.(1998)] N. Chiba, K. Muraoka et K. Fujita. *An erosion model based on velocity fields for the visual simulation of mountain scenery.* The Journal of Visualization and Computer Animation, tome 9, n° 6, p. 185–194, 1998.

[Christophe et al.(2008)] A. Christophe, V. Letort, I. Hummel, P.-H. Cournède, P. de Reffye et J. Lecoeur. *A model-based analysis of the dynamics of carbon balance at the whole plant level in Arabidopsis Thaliana.* Functional Plant Biology, tome 35(11), p. 1147–1162, 2008.

[Clark(1976)] J. H. Clark. *Hierarchical geometric models for visible surface algorithms.* Commun. ACM, tome 19, n° 10, p. 547–554, 1976.

[Costanza et Voinov(2004)] R. Costanza et A. Voinov. *Landscape Simulation Modeling : a Spatially Explicit, Dynamic Approach.* Modeling Dynamic Systems. Springer, 2004.

[Cournède et al.(2006)] P.-H. Cournède, M.-Z. Kang, A. Mathieu, J.-F. Barczi, H.-P. Yan, B.-G. Hu et P. de Reffye. *Structural Factorization of Plants to Compute their Functional and Architectural Growth.* Simulation, tome 82, n° 7, p. 427–438, 2006.

[Dachsbacher(2006)] C. Dachsbacher. *Interactive terrain rendering : towards realism with procedural models and graphics hardware.* Thèse de doctorat, University Erlangen-Nürnberg, Computer Sciences Institute, 2006. 162 p.

[de Reffye(1979)] P. de Reffye. *Modélisation de l'architecture des arbres par des processus stochastiques. Simulation spatiale des modèles tropicaux sous l'effet de la pesanteur. Application au Coffea robusta.* Thèse de doctorat, Université Paris-Sud, Centre d'Orsay, 1979.

[de Reffye et al.(1999)] P. de Reffye, F. Blaise, S. Chemouny, T. Fourcaud et F. Houllier. *Calibration of hydraulic growth model on the architecture of cotton plants.* Agronomie, tome 19, p. 265–280, 1999.

[de Reffye et al.(1988)] P. de Reffye, C. Edelin, J. Françon, M. Jaeger et C. Puech. *Plant models faithful to botanical structure and development.* Dans *Proc. SIGGRAPH 88, Computer Graphics*, tome 22(4), p. 151–158. 1988.

[de Reffye et Hu(2003)] P. de Reffye et B. Hu. *Relevant choices in botany and mathematics for building efficient dynamic plant growth models : the greenlab case.* Dans B. Hu et M. Jaeger (réds.), *Plant Growth Models and Applications*, p. 87–107. Tsinghua University Press and Springer, 2003.

[Deleuze(1996)] C. Deleuze. *Pour une dendrométrie fonctionnelle : essai sur l'intégration de connaissances écophysiologiques dans les modèles de production ligneuse.* Thèse de doctorat, Univeristé Claude Bernard - Lyon I, 1996.

[Deng et al.(2010)] Q. Deng, G. Yang, M. Jaeger et X. Zhang. *Multiresolution foliage for forest rendering.* Computer Animation and Virtual Worlds, tome 21, p. 1–23(23), 2010.

[Deussen et al.(1998)] O. Deussen, P. Hanrahan, B. Lintermann, R. Mĕ, M. Pharr et P. Prusinkiewicz. *Realistic modeling and rendering of plant ecosystems.* Dans

SIGGRAPH '98 : Proceedings of the 25th annual conference on Computer graphics and interactive techniques, p. 275–286. ACM Press, New York, NY, USA, 1998.

[Dong et al.(2008)] Q. Dong, G. Louarn, Y. Wang, J. Barczi et P. de Reffye. *Does the structure-fonction model greenlab deal with crop phenotypic plasticity induced by plant spacing ? a case study on tomato.* Annals of Botany, tome 101, n° 8, 2008.

[Dong et al.(2003)] Q. Dong, Y. Wang, J. Barczi, P. de Reffye et J. Hou. *Tomato growth modeling based on interaction of its structure-fonction.* Dans B. Hu et M. Jaeger (réds.), *Plant Growth Models and Applications PMA03*, p. 250–262. Tsinghua University Press and Springer (Beijing, China), 2003.

[Dunning et al.(1992)] J. B. Dunning, B. J. Danielson et H. R. Pulliam. *Ecological processes that affect populations in complex landscapes.* Oikos, tome 65, n° 1, p. 169–175, 1992.

[Ebert et al.(2002)] D. S. Ebert, K. F. Musgrave, D. Peachey, K. Perlin et S. Worley. *Texturing & Modeling : A Procedural Approach, Third Edition (The Morgan Kaufmann Series in Computer Graphics).* Morgan Kaufmann, 2002.

[Ervin et Hasbrouck(2001)] S. M. Ervin et H. H. Hasbrouck. *Landscape Modeling, Digital Techniques for Landscape Visualization.* McGraw-Hill, 2001.

[Ewen et al.(2000)] J. Ewen, G. Parkin et P. O'Connell. *SHETRAN : Distributed river basin flow and transport modelling system.* ASCE Journal of Hydrologic Engineering, tome 5, n° 1, p. 250–258, 2000.

[Fournier et al.(1982)] A. Fournier, D. Fussell et L. Carpenter. *Computer rendering of stochastic models.* Commun. ACM, tome 25, n° 6, p. 371–384, 1982.

[Gansner et North(1999)] E. R. Gansner et S. C. North. *An open graph visualization system and its applications to software engineering.* Software - Practice and Experience, tome 30, p. 1203–1233, 1999.

[Gilad et al.(2007)] E. Gilad, J. von Hardenberg, A. Provenzale, M. Shachak et E. Meron. *A mathematical model of plants as ecosystem engineers.* Journal of Theoretical Biology, tome 244, n° 4, p. 680 – 691, 2007.

[Godin et Caraglio(1998)] C. Godin et Y. Caraglio. *A multiscale model of plant topological structures.* Journal of Theoretical Biology, tome 191, p. 1–46, 1998.

[Greene(1989)] N. Greene. *Voxel space automata : modeling with stochastic growth processes in voxel space.* Dans *SIGGRAPH '89 : Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, p. 175–184. ACM Press, New York, NY, USA, 1989.

[Guo et al.(2007)] H. Guo, V. Letort, L. Hong, T. Fourcaud, P.-H. Cournède, Y. Lu et P. de Reffye. *Adaptation of the greenlab model for analyzing sink-source relationships in chinese pine saplings.* Dans T. Fourcaud et X. Zhang (réds.), *Plant growth Modeling, simulation, visualization and their Applications (PMA06).* IEEE Computer Society (Los Alamitos, California), 2007.

[Guo et al.(2006)] Y. Guo, Y. Ma, Z. Zhan, B. Li, M. Dingkuhn, D. Luquet et P. de Reffye. *Parameter optimization and field validation of the functional-structural model greenlab for maize.* Annals of Botany, tome 97, p. 217–230, 2006.

[Hallé et al.(1978)] F. Hallé, R. Oldeman et P. Tomlinson. *Tropical trees and forests, An architectural analysis*. Springer-Verlag, New-York, 1978.

[Hammes(2001)] J. Hammes. *Modeling of ecosystems as a data source for real-time terrain rendering*. Dans *DEM 2001, Digital Earth Moving First International Symposium*, tome 2181, p. 98–105. Springer Velag, LNCS, 2001.

[Heuvelink(1995)] E. Heuvelink. *Dry matter partitioning in a tomato plant : one common assimilate pool ?* Journal of Experimental Botany, tome 46, n° 289, p. 1025–1033, 1995.

[Howard(1994)] A. Howard. *A detachment limited model of drainage basin evolution*. Water Resources Research, tome 30, n° 7, p. 2261–2285, 1994.

[Jackson et Streator(2006)] R. L. Jackson et J. L. Streator. *A multi-scale model for contact between rough surfaces*. Wear, tome 261, n° 11-12, p. 1337 – 1347, 2006.

[Jaeger et de Reffye(1992)] M. Jaeger et P. de Reffye. *Basic concepts of computer simulation of plant growth*. Journal of Biosciences, tome 17, n° 3, p. 275–291, 1992.

[Jaeger et al.(2010)] M. Jaeger, R. Sun et V. Le Chevalier. *Efficient virtual plant data structure for visualization and animation*. Dans *IADIS 2010 : Computer Graphics, Visualization, Computer Vision and Image Processing*. 2010. En attente de publication.

[Jaeger et Teng(2003)] M. Jaeger et J. Teng. *Tree and plant volume imaging — an introductive study towards voxelized functional landscapes.* Dans J. M. et H. B.G. (réds.), *Proceedings PMA03 : 2003' International symposium on plant growth modeling, simulation, visualization and their application*, p. 169–181. Tsinghua University Press, Beijing, 2003.

[Jones(1992)] H. Jones. *Plants and Microclimate*. Cambridge University Press, 1992.

[Kang et al.(2004)] M.-Z. Kang, P.-H. Cournède, J. Le Roux, P. de Reffye et B.-G. Hu. *Theoretical study and numerical simulation of a stochastic model for plant growth*. Dans *CARI04, Tunisia*. 2004.

[Kaufmann(1991)] A. Kaufmann (réd.). *Volume Vizualisation*. IEEE Computer Society Press, 1991.

[Knuth(1984)] D. E. Knuth. *Literate programming*. Comput. J., tome 27, n° 2, p. 97–111, 1984.

[Kosugi et al.(2002)] K. Kosugi, J. Hopmans et J. Dane. *Water Retention and Storage - Parametric Models*, tome 5 de *Methods of Soil Analysis*, chapitre 4. Physical Methods, p. 739–758. Soil Science Society of America, 2002.

[Le Chevalier et Jaeger(2010)] V. Le Chevalier et M. Jaeger. *A bottom-up approach of landscape simulation leading to a generic synchronization formalism and competition model*. Dans *Proceedings of LandMod 2010 - International Conference on Integrative Landscape Modelling*. 2010. En attente de publication.

[Le Chevalier et al.(2009)] V. Le Chevalier, M. Jaeger et P.-H. Cournède. *Synchronisation formalism, resource and plant models for plant ecosystem simulation*. Dans

B. Li, M. Jaeger et Y. Guo (réds.), *International Symposium on Plant Growth Modeling and Applications*, p. 277–284. IEEE Computer Society, Los Alamitos, CA, USA, 2009.

[Le Chevalier et al.(2007a)] V. Le Chevalier, M. Jaeger, X. Mei et P.-H. Cournède. *Simulation and visualisation of functional landscapes : Effects of the water resource competition between plants.* Journal of Computer Sciences and Technology, tome 22, n° 6, p. 835–845, 2007a.

[Le Chevalier et al.(2007b)] V. Le Chevalier, M. Jaeger, X. Mei, A. Lesluye et P. Cournède. *A functional landscape prototype to simulate water resource competition between plants.* Dans T. Fourcaud et X. Zhang (réds.), *PMA06 - Plant growth Modeling, simulation, visualization and their Applications*, p. 124–131. IEEE Computer Society, Los Alamitos, California, 2007b.

[Letort(2008)] V. Letort. *Multi-scale analysis of source-sink relationships in plant growth models for parameter identification. Case of the GreenLab model.* Thèse de doctorat, Ecole Centrale Paris, 2008.

[Letort et al.(2008)] V. Letort, P. Mahe, P.-H. Cournède, P. de Reffye et B. Courtois. *Quantitative genetics and functional-structural plant growth models : Simulation of quantitative trait loci detection for model parameters and application to potential yield optimization.* Annals of Botany, tome 101, n° 8, 2008.

[Li et al.(2009)] Z. Li, V. Le Chevalier et P.-H. Cournède. *Towards a continuous approach of functional-structural plant growth.* Dans B. Li, M. Jaeger et Y. Guo (réds.), *International Symposium on Plant Growth Modeling and Applications*, p. 334–340. IEEE Computer Society, Los Alamitos, CA, USA, 2009.

[Liu et al.(2006)] Y. Liu, S. Gebremeskel, F. De Smedt, L. Hoffmann et L. Pfister. *Predicting storm runoff from different land use classes using a gis-based distributed model.* Hydrological Processes, tome 20, n° 6, p. 533–548, 2006.

[Ma et al.(2008)] Y. Ma, M. Wen, Y. Guo, B. Li, P.-H. Cournède et P. de Reffye. *Parameter optimization and field validation of the functional-structural model greenlab for maize at different population densities.* Annals of Botany, tome 101(8), 2008.

[Mailhol et al.(1997)] J. Mailhol, A. Olufayo et P. Ruelle. *Sorghum and sunflower evapotranspiration and yield from simulated leaf area index.* Agri. Water Manag., tome 35, p. 167–182, 1997.

[Mathieu(2006)] A. Mathieu. *Essai sur la modélisation des interactions entre la croissance d'une plante et son développement dans le modèle GreenLab.* Thèse de doctorat, Ecole Centrale Paris, 2006.

[Mathieu et al.(2009)] A. Mathieu, P.-H. Cournède, V. Letort, D. Barthélémy et P. de Reffye. *A dynamic model of plant growth with interactions between development and functional mechanisms to study plant structural plasticity related to trophic competition.* Annals of Botany, 2009.

[Max(1987)] J. Max. *Méthodes et techniques de traitement du signal et applications aux mesures physiques.* Masson, 1987. 4ème édition.

[Mayer et Sarjoughian(2007)] G. R. Mayer et H. S. Sarjoughian. *Complexities of simulating a hybrid agent-landscape model using multi-formalism composability*. Dans *SpringSim '07 : Proceedings of the 2007 spring simulation multiconference*, p. 161–168. Society for Computer Simulation International, San Diego, CA, USA, 2007.

[Mech et Prusinkiewicz(1996)] R. Mech et P. Prusinkiewicz. *Visual models of plant interacting with their environment*. Dans *ACM SIGGRAPH'96 Proceedings*, p. 397–410. 1996.

[Mitasova et al.(1996)] H. Mitasova, L. Mitas, W. M. Brown et D. Jonhston. *Multidimensionnal soil erosion/deposition modeling*. Rapport annuel, Geographic Modeling and Systems Laboratory, 1996. Part III.

[Musgrave et al.(1989)] F. Musgrave, C. Kolb et R. Mace. *The synthesis and rendering of eroded fractal terrains*. Computer Graphics, tome 23, n° 3, p. 41–50, 1989.

[Neidhold et al.(2005)] B. Neidhold, M. Wacker et O. Deussen. *Interactive physically based fluid and erosion simulation*. Dans *Eurographics Workshop on Natural Phenomena, Dublin*, p. 25–32. Eurographics, 2005.

[Pallas et al.(2009)] B. Pallas, C. Loi, A. Christophe, P.-H. Cournède et J. Lecœur. *A stochastic growth model of grapevine with full interaction between environment, trophic competition and plant development*. Dans B. Li, M. Jaeger et Y. Guo (réds.), *International Symposium on Plant Growth Modeling and Applications*, p. 95–102. IEEE Computer Society, Los Alamitos, CA, USA, 2009.

[Prusinkiewicz(2004)] P. Prusinkiewicz. *Modeling plant growth and development*. Current opinion in plant biology, tome 7, n° 1, p. 79–84, 2004.

[Quesnel et al.(2009)] G. Quesnel, R. Duboz et E. Ramat. *The Virtual Laboratory Environment – An operational framework for multi-modelling, simulation and analysis of complex dynamical systems*. Simulation Modelling Practice and Theory, tome 17, p. 641–653, 2009.

[Richards(1931)] L. A. Richards. *Capillary conduction of liquids through porous mediums*. Physics, tome 1, n° 5, p. 318–333, 1931.

[Schneider et al.(2006)] J. Schneider, T. Boldte et R. Westermann. *Real-time editing, synthesis, and rendering of infinite landscapes on GPUs*. Dans *Vision, Modeling and Visualization*. Aachen, Germany, 2006.

[Sievänen et al.(2000)] R. Sievänen, E. Nikinmaa, P. Nygren, H. Ozier-Lafontaine, J. Perttunen et H. Hakula. *Components of a functional-structural tree model*. Annals of Forest Sciences, tome 57, p. 399–412, 2000.

[Taddei et al.(2000)] U. Taddei, O. David et C. Michl. *Application of VisAD in hydrological modeling and simulation*. SIGGRAPH Comput. Graph., tome 34, n° 1, p. 48–50, 2000.

[Tarboton(1997)] D. Tarboton. *A new method for the determination of flow direction and upslope areas in grid digital elevation model*. Water Resources Research, tome 33, n° 2, p. 309–319, 1997.

[Tucker et al.(1997)] G. Tucker, N. Gasparini et S. Lancaster. *An integrated hillslope and channel evolution model as an investigation and prediction tool.* Annual report, Child model, 1997.

[Turc(1961)] L. Turc. *Évaluation des besoins en eau d'irrigation, évapotranspiration potentielle, formule simplifiée et mise à jour.* Annales agronomiques, tome 12, n° 1, p. 13–49, 1961.

[Turner et al.(1989)] M. G. Turner, V. H. Dale et R. H. Gardner. *Predicting across scales : Theory development and testing.* Landscape Ecology, tome 3, n° 3/4, p. 245 – 252, 1989.

[Varado et al.(2005)] N. Varado, I. Braud et P. Ross. *Development and assessment of an efficient vadose zone module solving the 1D Richards' equation and including root extraction by plants.* Journal of Hydrology, 2005.

[Varenne(2003)] F. Varenne. *La simulation informatique face à la méthode des modèles. le cas de la croissance des plantes.* Nature Sciences Sociétés, tome 11, p. 16–28, 2003.

[Vrugt et al.(2001)] J. Vrugt, J. Hopmans et J. Simunek. *Calibration of a two-dimensional root water uptake model.* Soil Science Society of America Journal, tome 65, n° 4, p. 1027–1037, 2001.

[Wang et Hu(2009)] N. Wang et B.-G. Hu. *Aeolian sand movement and interacting with vegetation : A gpu based simulation and visualization method.* Dans B. Li, M. Jaeger et Y. Guo (réds.), *International Symposium on Plant Growth Modeling and Applications*, p. 401–408. IEEE Computer Society, Los Alamitos, CA, USA, 2009.

[White(1979)] J. White. *The plant as a metapopulation.* Annu. Rev. Ecol. Syst, tome 10, p. 109–145, 1979.

[Wiens(1989)] J. A. Wiens. *Spatial scaling in ecology.* Functional Ecology, tome 3, n° 4, p. 385–397, 1989.

[Witelski(2005)] T. Witelski. *Motion of wetting fronts moving into partially pre-wet soil.* Advances in Water Resources, tome 28, p. 1133–1141, 2005.

[Wu et al.(2005)] L. Wu, F.-X. Le Dimet, B. Hu, P.-H. Cournède et P. de Reffye. *A water supply optimization problem for plant growth based on greenlab model.* ARIMA Journal, p. 194–207, 2005.

[Yan et al.(2004)] H. Yan, M.-Z. Kang, P. de Reffye et M. Dingkuhn. *A dynamic, architectural plant model simulating resource-dependent growth.* Annals of Botany, tome 93, p. 591–602, 2004.

[Zeigler et Vahie(1993)] B. Zeigler et S. Vahie. *Devs formalism and methodology : unity of conception/diversity of application.* Dans *Proceedings of the 25th Winter Simulation Conference*, p. 573–579. ACM Press, 1993.

[Zeigler et al.(2000)] B. P. Zeigler, H. Praehofer et T. G. Kim. *Theory of Modeling and Simulation.* Academic Press, 2000.

[Zhou et al.(2007)] H. Zhou, J. Sun, G. Turk et J. Rehg. *Terrain synthesis from digital elevation models.* IEEE Transactions on Visualization and Computer Graphics, tome 13, n° 4, p. 834–848, 2007.